MITRE-Bedford

MTR-4723
VOL. II

JSC #14793

OCT 1 - 1979

TMS Communications Software Volume II- Bus Interface Unit

Paul J. Gregor

APRIL 1979

MITRE

MITRE Technical Report
MTR-4723
Vol. II

TMS Communications Software
Volume II- Bus Interface Unit

Paul J. Gregor

APRIL 1979

CONTRACT SPONSOR NASA/JSC
CONTRACT NO. F19628-79-C-0001 T5295F
PROJECT NO. 8470
DEPT. D72

THE MITRE CORPORATION
HOUSTON, TEXAS

Department Approval: Edwin S Hendon

MITRE Project Approval: Edwin S Hendon

ABSTRACT

MITRE has installed a prototype data bus communication system at NASA's Johnson Space Center. The system supports the space shuttle's Trend Monitoring System (TMS) and also provides a basis for evaluation of the bus concept. Installation of the system included developing both hardware and software interfaces between the bus and the specific TMS computers and terminals. The software written for the microprocessor-based bus interface units supplied by MITRE is described in this paper. The software implements both the general bus communications protocol and also the specific interface protocols for the TMS computers and terminals.

TABLE OF CONTENTS

TABLE OF CONTENTS

(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Concluded)

LIST OF ILLUSTRATIONS

LIST OF TABLES

TMS COMMUNICATIONS SOFTWARE
VOLUME II - BUS INTERFACE UNIT

SECTION I
BACKGROUND

1.0 HISTORY OF TMS

The Orbiter Data Reduction Complex (ODRC) at NASA's Johnson Space Center has the responsibility of providing data reduction for measurements collected during manned spaceflight missions. This data reduction involves the extraction of requested data from magnetic tapes, the calibration of the raw measurements and the conversion of the measurements to engineering units, and the display of the data in any of a variety of output forms. Ordinarily, the work of the ODRC is done in response to written requests and has a planned turnaround time ranging from several hours to several days, depending on the priority of the request.

In 1977, however, as data processing requirements for Operational Flight Tests (OFT) of the Space Shuttle were being considered, it was established that NASA/JSC's Structures and Mechanics Division (SMD) needed to view thermal parameters for the shuttle in near real time. As a consequence, the Institutional Data Systems Division (IDSD), which is responsible for the ODRC, chose to implement an interactive graphics system to display plots of current, projected, and historical thermal data for the Shuttle. The system, termed the Trend Monitoring System (TMS), was implemented by IDSD's Engineering and Special Development Branch (FD7) using a MODCOMP IV/35 host minicomputer and MEGATEK 5000 intelligent graphics terminals (based around Data General NOVA/3 minicomputers).

In the TMS, the terminals and the host computer are separated by a distance of about 1600 feet, and the requirements for response time dictate that a high data rate be provided on the communications path between the host and the terminals. Conventional communications systems to meet these

requirements are not readily available. MITRE has developed a coaxial cable bus communications system which provides a communications bandwidth of up to 307.2 Kbps over a distance of several miles. IDSD consequently elected to install a prototype bus communications system with the dual objective of supporting the TMS needs and of providing a test bed for further evaluation of the bus concept's ability to meet digital computer communication needs.

1.1 Basic Structure of the TMS Network

The TMS consists of the communications cable, subscriber devices (the MODCOMP, the MEGATEK/NOVA terminals, and serial terminals such as Texas Instruments Silent 700), and devices which interface the subscribers to the cable (Bus Interface Units - BIUs). Subscriber devices may or may not have the capability to execute a computer program to interface with a BIU. In the TMS both the MODCOMP and the MEGATEK interface to a BIU through computers. This paper is concerned only with BIU software to support these computer-based interfaces.

The components of the TMS communication interface are shown in Figure 1.1-1. As the figure indicates, an applications program executing in a subscriber device (e.g., the terminal graphics program described in [1]) must send data through several levels of hardware and software within the subscriber device before it is passed on to the BIU. Some of these levels are standard in that they are used for I/O to a variety of devices. Others were developed specifically for TMS (and are described in [2] and [3]). For simplicity of presentation in this paper, however, all subscriber device levels will be collectively referred to as the subscriber device, with no distinction made between hardware and software.

BIUs are also composed of subsystems. The BIU contains both software (the focus of this paper) and hardware. The hardware consists of a microprocessor (MOS Technology Model 6502A) plus devices for interfacing to the cable communications network and to the subscriber device.

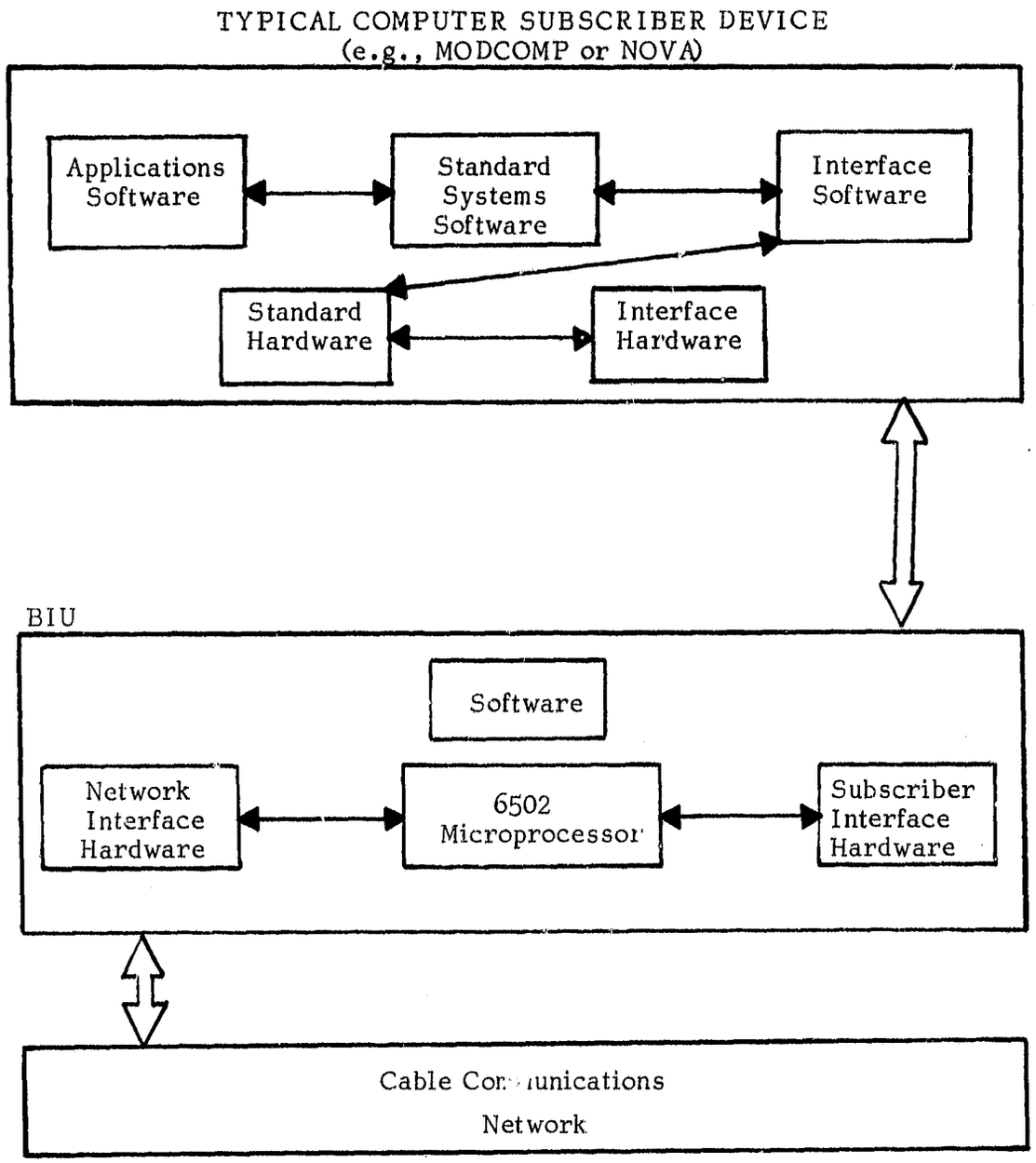


Figure 1.1-1

Components and Subsystems of the TMS Bus System

BIU hardware is described in detail in [4]. A brief summary of BIU hardware is provided in this paper in Section III to aid in understanding the structure of the software.

Different "types" of BIUs are required to interface to different types of subscriber devices. The basic differences between the "types" of BIU are in software. As will be detailed later, there are five different versions of the BIU software. There are only two different versions of BIU hardware, but these two versions differ only in one hardware component - the external chassis. One version of the external chassis provides a plug for a parallel interface to the subscriber, the other version provides a serial RS-232C interface.

1.2 Overview of This Report

The focus of this report is the TMS BIU software for the MODCOMP BIU and for the NOVA BIU. The report is primarily intended to be a reference manual describing the functions performed by the BIU and how it performs them. The primary intended audience is the systems programmer charged with maintaining the BIU software.

The report is structured into two parts. The first part (Sections II through IV) briefly describes the nature and function of the TMS BIU software. This part is intended for general reading in a sequential manner.

Sections V through XI form the second part of the report and carry the title "BIU Software Maintenance Notebook." The various sections focus in on different aspects of the software code. The sections are intended as references useful in discovering particular categories of problems that may develop in maintaining or altering the BIU software.

PART I
NATURE AND FUNCTION OF
TMS BIU SOFTWARE
FOR THE
MODCOMP AND MEGATEK INTERFACES

SECTION II
GENERAL DESCRIPTION OF BIU SOFTWARE

2.0 PURPOSE

The basic purpose of a BIU is to enable a subscriber device to send messages onto the network and to receive messages from it. To achieve this end, a BIU must cooperate with the subscriber device's interface hardware and software. As interface characteristics vary among subscriber devices, so too must there be variations among the corresponding BIUs. The use of software for implementing those interface functions which vary among subscriber devices allows some standardization of the hardware in the BIU. Specific interface functions performed by the BIU include:

- reading, interpreting, and altering status and control lines connecting the BIU to the subscriber device,
- converting bit-parallel transmissions from the subscriber to a bit-serial transmission on the network and vice versa), and
- segmenting messages received from the subscriber device which exceed network standards, or segmenting messages received from the network which exceed device capabilities.

The BIU software also performs functions other than handling strictly device-dependent interfaces. These functions can be placed under the general title of message transmission quality assurance functions. Specific functions performed include:

- gatekeeping; controlling the flow of messages into and out of the subscriber device; guaranteeing that the subscriber device receives only messages

intended for it; and only messages which it is, in some sense, expecting; guaranteeing that messages sent onto the network are directed to the correct destination*;

- ensuring that a message is transmitted and received via the network without error through the use of acknowledgement protocols, parity codes, and retransmission capabilities;
- buffering messages to ensure that they are not lost, and to avoid over-burdening the subscriber device; accepting and temporarily holding messages received from the network but which the subscriber device is not yet ready to handle (or received from the device when the net is busy); and
- monitoring the performance of the network to aid in the maintenance of the system; keeping statistics on transmission errors and transmissions attempted.

2.1 Past and Current Versions

As mentioned above, different characteristics are required in BIUs interfacing to different devices. In the TMS system there are five different uses for a BIU.

1) the MODCOMP BIU - interfaces the host processor to the network;
2) the NOVA BIU - interfaces a graphics terminal to the network;

---

*In the TMS MODCOMP system, this latter aspect of gatekeeping is shared by the BIU software with the interface software in the MODCOMP.

3) the Serial Terminal BIU - interfaces an ordinary RS232 terminal such as a Texas Instruments Silent 700 to the network;

4) the Backboard BIU - a stand-alone BIU which returns to the sender any messages it receives; and

5) the Listener BIU - a BIU which monitors all traffic on the net and prints user-selected portions on its attached terminal.

Corresponding to these five users of the BIU are five versions of BIU software.* The fourth and fifth versions of the software (Backboard and Listener versions) are used primarily for system maintenance. The serial-terminal BIU is available to the TMS user, but is not a major component of the system. The code for the Backboard and Serial Terminal BIU is relatively straightforward and will thus not be discussed further in this paper. The Listener is described in [5]. The focus of this paper will thus be on the MODCOMP and NOVA versions of the BIU software.

The MODCOMP and NOVA versions of the software have both large differences and basic similarities. To better understand these differences and similarities, a few comments on the origins of the MODCOMP and NOVA versions may be helpful.

Both the MODCOMP and NOVA BIU software versions are descendents of earlier versions of BIU software designed for other subscriber devices. The basic structure of these earlier versions included one routine for taking data from the subscriber, one for sending data to the subscriber, one for taking data from the net, one for giving data to the net, and various support routines.

*There are, however, only two versions of the BIU hardware. The MODCOMP and NOVA software versions are used with a parallel interface BIU hardware unit and the serial terminal and Listener software versions are used with a serial interface BIU hardware unit. The Backboard software can be used with either type of hardware.

In the TMS development, however, new computer-to-BIU protocols were developed for both the MODCOMP and the NOVA [3]. At the MODCOMP, existing hardware was employed, while at the NOVA new interface hardware was designed and built. Hardware documentation uses different names for similar signals at the two interfaces, and such differences as this are inevitably reflected in the BIU code.

The result of the development process is that the NOVA and MODCOMP BIU software versions exhibit

- similarities in network interfacing routines and utility routines,
- differences due to the differences in subscriber interface protocols,
- similarities because both use the same BIU hardware to interface to the subscriber,
- differences resulting from hardware signal names (reflected in BIU software variable names) and from design team members.

Unfortunately, this last category of differences obscures some of the similarities between the two versions.

2.2 General Structure

In general the software was developed following the principle of modularity. There is thus a fairly large number of subroutines. The software for the MODCOMP BIU consists of 21 routines; the NOVA version includes 23 routines. Fifteen of these routines appear in both versions, with some difference between versions of the routines. The principle of modularity was, however, not uniformly applied. A number of the routines are fairly short (5-15 instructions), but some are relatively long (100-140 instructions).

Part of the motivation for modularity was the production of easy-to-read code. Thus, in one case, a large section of code was divided up into two routines, even though the second routine was only called from one place in the first routine and always returned to the calling location. The PUTM and WRITEM routines (see Section X) are the case in point.

A second motivation for modularity was the conservation of PROM storage. Generally, wherever a series of instructions was found to be repeated in more than one location, these instructions were made into a subroutine. Because of the limited PROM storage (relative to the logic required) in the NOVA version, subroutines were used even if only several bytes could be saved at the expense of a subroutine call. Though PROM space was not as serious a problem in the MODCOMP version, the same subroutine structure was implemented there to maintain as much similarity as possible between the two versions.

The 21 or 23 routines of the two versions can be grouped in several ways. Table 2.2-I divides the routines into subscriber interface routines, network interface routines, and general coordination and support routines. Alternatively, the routines can be divided by method of invocation as indicated in Table 2.2.II. Three routines service interrupts, one routine is a short general sequencer of activities which repeatedly calls the processing routines, some routines are directly called by this "main-loop" sequencer, and other routines are general support routines. The exact functions of the routines will be considered in more detail later in this paper.

2.3 Language and Programming Practices

All BIU software is coded in assembly language. Assembly language was used not only because of its efficiencies, but also because a high-order language was not available for use with the chosen microprocessor. A later chapter in this paper discusses some unusual characteristics of the 6502 assembly language.

Table 2.2-I

BIU Software Subroutines Classified by General Function

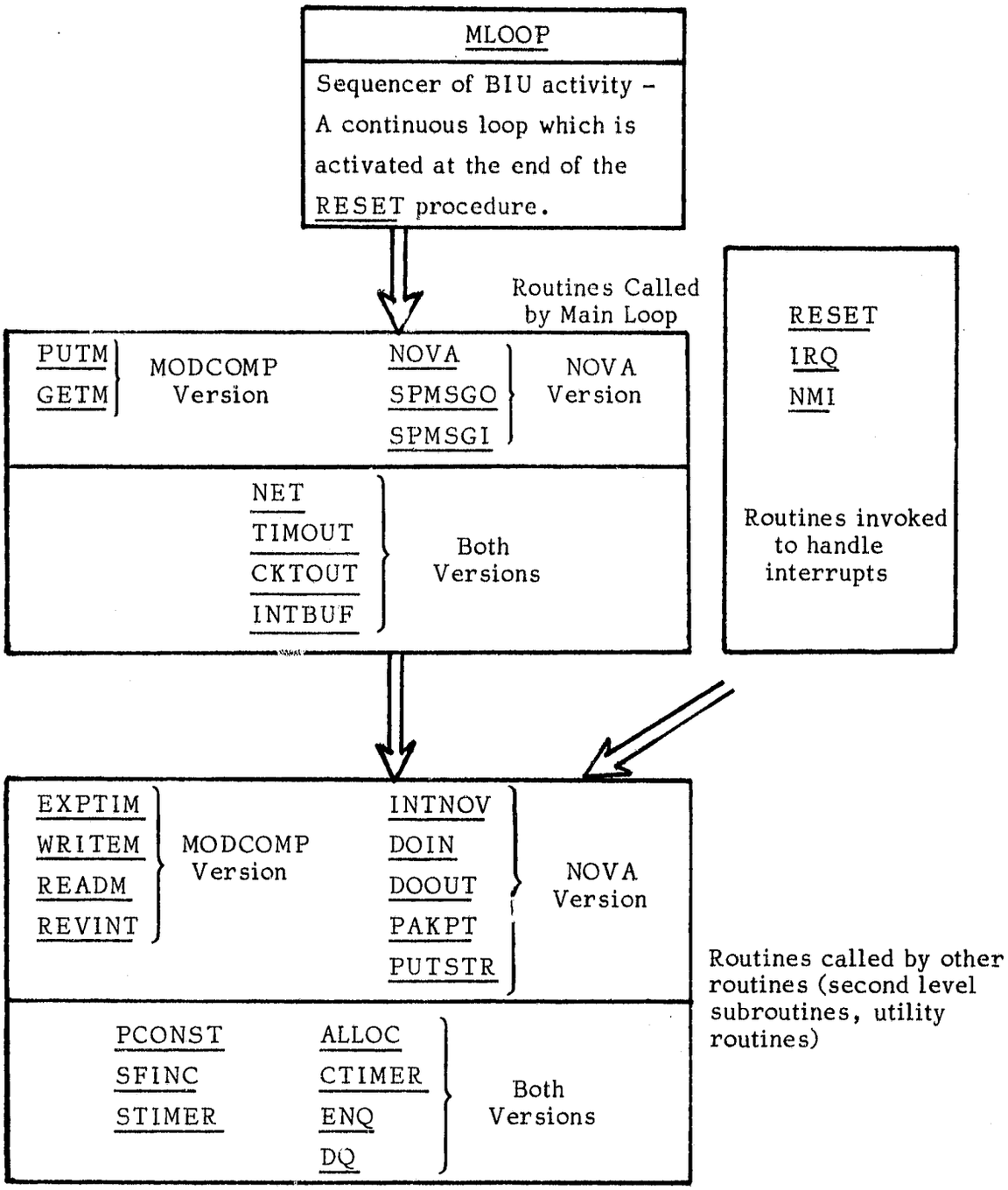
SUBSCRIBER INTERFACE ROUTINES	
MODCOMP	NOVA
PUTM	NOVA
GETM	INTNOV
WRITEM	DOIN
READM	DOOUT
REVINT	PAKPT
EXPTIM	SPMSGO
	SPMSGI
	PUTSTR

NETWORK INTERFACE ROUTINES
NET
IRQ
NMI
INTBUF

GENERAL COORDINATION & SUPPORT SUBROUTINES (Both Versions)		
RESET	PCONST	ALLOW
MLOOP	STIMER	ENQ
TIMOUT	CTIMER	DQ
CKTOUT	SFINC	

Table 2.2-II

BIU Software Subroutines Classified by Method of Invocation



The code for both the MODCOMP and NOVA versions [6] has been liberally interspersed with comments. Most of what is said in this document is included in a source code listing, though perhaps in a different form. Some aspects of the software undoubtedly will have been left unexplained. This may be due to oversight or to the fact that the original reasoning behind the code has long since been forgotten. The reader's (or maintenance programmer's) indulgence is begged in these cases.

Early versions of both the MODCOMP and BIU software generally labelled statements with a simple letter-number combination (e.g., NE10). It was found that maintaining this code was difficult because one had to closely read a sequence of instructions to determine what functions were being performed. During a cleanup effort, many of these labels were changed to more meaningful phrases (e.g., WAIT, PREPAR, etc.).

SECTION III
GENERAL DESCRIPTION OF BIU HARDWARE

3.0 INTRODUCTION

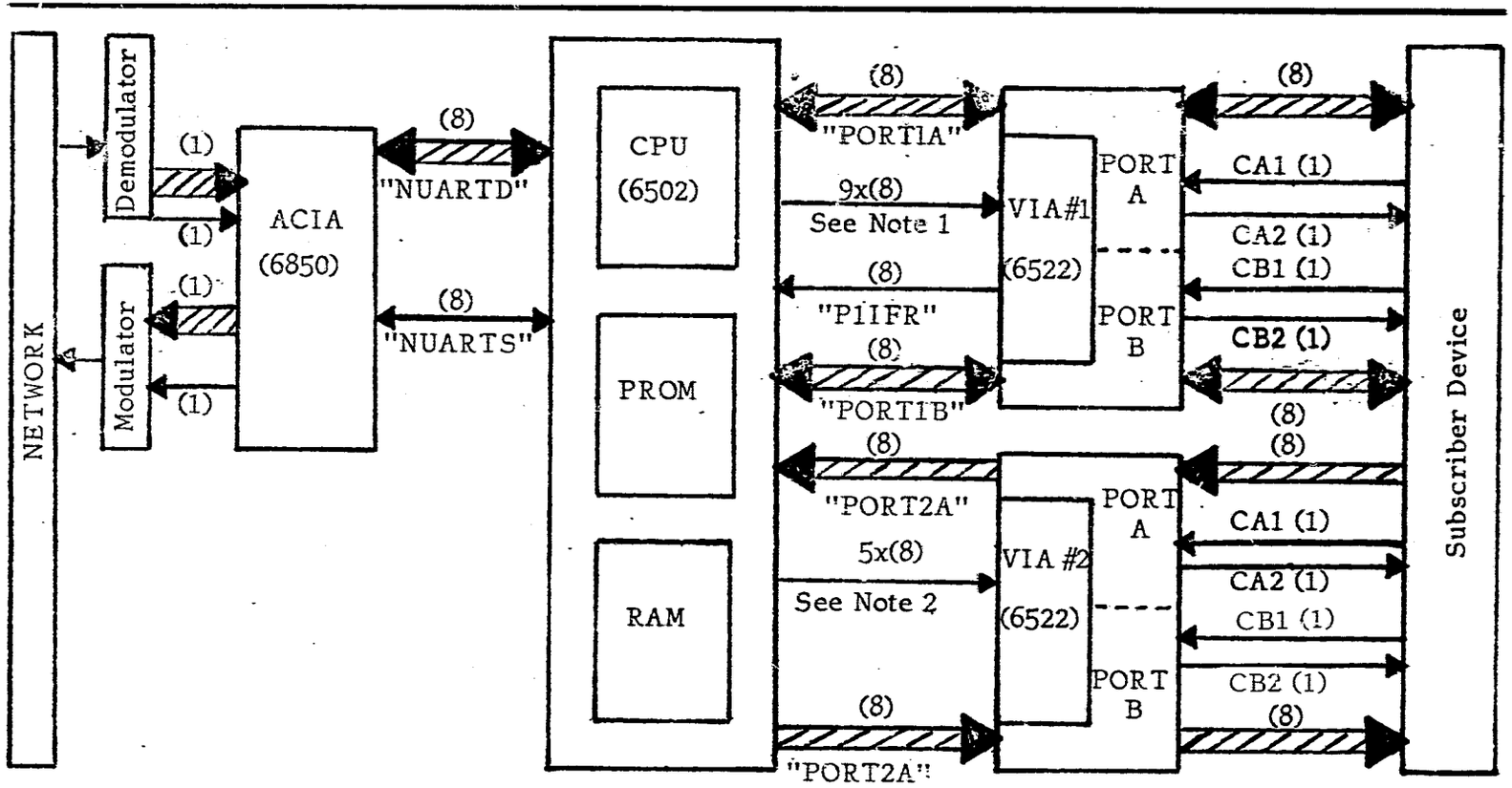
The main hardware components of the parallel-interface BIUs used in the TMS are shown in Figure 3.0-1. The following paragraphs describe briefly the function of each component and indicate how each affects the flow of data through the BIU. A detailed description of the BIU hardware is contained in [4].

3.1 6502 Microprocessor

The heart of the BIU is the model 6502 microprocessor made by MOS Technology, Inc. The 6502, by executing the software contained in the PROM, controls the interchange of data between the net and the subscriber device and implements the quality assurance functions indicated in the previous chapter. General characteristics of the 6502 are that it has:

- six registers (three special purpose - a Program Counter, a Stack Pointer, and a Processor Status Register; and three available for program use - an Accumulator and X and Y Index Registers),
- fifty-six instructions and thirteen addressing modes,
- a 600 nanosecond instruction cycle,
- three separate interrupt lines, and
- a hardware-maintained "stack."

As shown in Figure 3.0-1, the 6502 is interfaced to the network through an Asynchronous Communications Interface Adapter (ACIA) and to the subscriber device through two Versatile Interface Adapters (VIAs). The activity of the 6502 (i.e., the selection of portions of the software to be executed) is determined both by interrupts from these interface devices



"NAME" = Variable name used in software (see Part II of paper)	Notes:
////// = data path	1. Nine eight-bit registers: "P1ADDR", P1BDDR", "T1MRL", "T1MRH", "T1M2L", "T1M2H", "P1ACR", "P1PCR", "P1IER".
——— = control or status signal path	2. Five eight-bit registers: "P2ADDR", "P2BDDR", "P2ACR", "P2PCR", "P2IER"
(n) = path width in bits	

Figure 3.0-1 BIU Hardware Component Structure

and by the state of status signals in these devices monitored by the 6502. The meanings of the various interrupts and status signals will be discussed in Part II of this paper.

All transfers between the subscriber device and the network must go through the 6502. For instance, in sending a message from the subscriber device onto the net, the 6502 must explicitly read each 16-bit word from the VIA into buffer storage in RAM, "process" the message, and then write the message one byte at a time to the ACIA (and hence to the net) from buffer storage.

The "stack" maintained by the 6502 is used primarily in establishing subroutine linkages. Usage of the stack is discussed in Section V.

For more information on the 6502, see [7].

3.2 PROM

The BIU has a total of 2048 bytes of erasable, programmable read-only memory on two chips. These chips contain the software (instructions and constants) described in the remainder of this document. The size of the PROMs has been somewhat of a limiting factor in the development of the software. Currently, the MODCOMP version of the software uses all but 210 of the 2048 bytes and the MEGATEK version uses all but 7.

3.3 RAM

The BIU has 3072 bytes of random access memory which can be both read and written. Some of this memory is used to hold the "stack" maintained and referenced by certain 6502 instructions (see Section V). A 128-byte portion of RAM has been reserved by convention for the "stack" (see Section V for more on space allocation for the "stack.") The remainder of RAM is used for various pointers, counters, and indices (approximately 145 bytes) and for buffering messages being passed between the subscriber device and the net (21 buffers each 128 bytes long).

3.4 ACIA

A MOTOROLA mode' 6805 Asynchronous Communications Interface Adapter (ACIA) is used to interface the BIU to the network cable. The single-chip ACIA acts like a peripheral controller under the direction of the 6502 microprocessor. The basic functions of the ACIA are either to take a byte of data from the 6502 and transmit it bit-serially onto the net, or to accumulate a byte of data from the net and make it available to the 6502. The ACIA may also be queried by the 6502 as to whether the network cable is in use at a particular time or not.

The 6502 controls, monitors, and exchanges data with the ACIA by directly accessing the registers internal to the ACIA. As indicated in Figure 3.0-1, the ACIA has a bit-serial connection to the network cable and eight-bit parallel data and control links to the 6502. The ACIA can also trigger two different types of interrupts to the 6502. The interface protocol between the ACIA and the 6502 is discussed in detail in Section X. For more information on the ACIA itself please see [8].

3.5 VIA

Two Versatile Interface Adapters (Model 6522 from MOS Technology, Inc.) are used to provide a 16-bit parallel interface between the BIU and the subscriber device. Though the two VIAs are identical chips and are wired to the 6502 in a similar manner, they perform unique functions and are connected to dissimilar portions of the subscriber device. Each VIA has two eight-bit sets of lines connecting to the subscriber device. One VIA is used to transfer data between the 6502 and the subscriber device in a 16-bit-parallel manner. The 16 lines of the other VIA are used as status flags: eight are available for telling the 6502 the status of the subscriber device and any on going transfer, eight are available for telling the subscriber device the status of the 6502 and any ongoing transfer (not all 16 lines are currently used).

The 6502 controls, monitors, and exchanges data with the VIA by directly accessing the registers internal to VIA. Each VIA has two eight-bit data registers and twelve addressable control registers. Each VIA interfaces to the subscriber device through two eight-bit sets of data lines and four auxiliary "handshaking" lines for coordinating the transfers. The interface protocol between the 6502, the VIA, and the subscriber device is described in detail in Section X. For more information on the VIA itself see [9].

3.6 Address Space of the 6502

As mentioned above the 6502 controls and monitors the VIAs and the ACIA by directly accessing their internal control registers. In the BIU these registers are connected to the 6502 in such a way that they appear to the BIU software to be ordinary memory locations. For example, a store of the value "-1" into location $0C00_{16}$ by the 6502 will change the value of the ACIA's control register to "-1." Thus, the BIU software contains no explicit I/O instructions, but uses instead sequences of LOAD and STORE instructions. Figure 3.6-1 indicates the specific 6502 addresses connected to the VIAs, the ACIA, PROM, and RAM.

HEXADECIMAL ADDRESS	USAGE
0000 ⌇ 0090 (approximately)	RAM - Pointers, counters, flags, etc.
0091 ⌇ 00FF	RAM - Unused
0100 ⌇ 017F	RAM - Stack storage
0180 ⌇ 0BFF	RAM - Buffer Storage
0C00 0C01	Network ACIA - Control & Data Registers
⌇	Unused (no memory present for these addresses)
1010 ⌇ 101F	Data Transfer VIA - Control & Data Registers
1020 ⌇ 102F	Status Transfer VIA - Control & Data Registers
⌇	Unused (no memory present for these addresses)
1400 ⌇ 1401	Subscriber ACIA - Control & Data Registers
⌇	Unused (no memory present for these addresses)
F800 ⌇ FFFF	PROM - Software Instructions & Constant Tables

Figure 3.6-1
Map of the Address Space of the BIU's 6502 Microprocessor

SECTION IV
FUNCTIONAL CHARACTERISTICS OF THE BIU SOFTWARE

4.0 INTRODUCTION

Section 2.0 identified a series of seven functions that are performed in the BIU software. The first three functions centered on low-level interface protocols. Explanations of these protocols require presentation of considerable detail of both the hardware and software. These explanations are thus reserved for Part II of this paper.

The quality assurance functions, however, are based on a high-level protocol more suitable to a simple presentation. The following paragraphs describe the procedures used to implement these functions in the BIU software.

4.1 Gatekeeping

One of the purposes of the BIU is to control the flow of messages into and out of the subscriber device. The BIU attempts to guarantee that the subscriber device receives only messages intended for it, and only messages which it is, in some sense, expecting.

To help accomplish the gatekeeping functions, each BIU is assigned an address. Each BIU may have more than one address assigned to it, but each address is assigned to only one BIU. Messages sent on the network are transmitted in packets of bits which are accessible to all BIUs on the net. Each message packet, however, has an attached header, one field of which contains the address of the destination BIU. Each BIU on the network reads the header of any message packet transmitted. Each BIU will however read only these messages with a matching destination address. This accomplishes the first phase of gatekeeping.

Unexpected messages are screened from the subscriber in one sense in that no data is sent to the subscriber until it initiates a read for data from the BIU. A second level of screening is introduced in the NOVA version of the software. To function properly the NOVA must have been loaded with an executable program either from disk or from the MODCOMP through the net. The BIU is capable of sensing whether the NOVA has been bootstrap-loaded ("booted") and whether an incoming packet is a message packet or a portion of a program being booted. If the state of the NOVA and the type of the incoming packet are not compatible, the packet is simply ignored. A more detailed discussion of the process is found in [3].

4.2 Transmission Quality Assurance

Several procedures are implemented in the BIU software to guarantee that a message is correctly received by the destination BIU. These procedures are:

- detection of the presence of traffic on the network to avoid generating a message that will collide with the transmission of another BIU,
- detection of the occurrence of a collision between two messages,
- transmissions of acknowledgements of correct reception of a packet and retransmission if the acknowledgement is not received, and
- generating and processing two types of parity.

The following paragraphs describe each of these procedures.

4.2.1 Collision Avoidance and Detection

The network can successfully transmit only one message at a time. Messages will be garbled if two BIUs attempt simultaneously to send messages on the net. To avoid such a situation each BIU can sense whether there is any traffic on the network. If there is no traffic, the BIU assumes it is safe to transmit and begins to do so.

This procedure reduces, but does not eliminate, the possibility of simultaneous transmissions. Two BIUs could have queried the network status at approximately the same time and both concluded that it was safe to transmit. The resulting two messages will garble each other. The BIU uses a "listen-while-talk" protocol to detect such a collision of messages.

The data transmitted from a BIU will be received by the BIU a short (but finite) amount of time later, allowing for the propagation delays of the cable. Each BIU can thus detect whether a collision has occurred between its transmission and that of another BIU by comparing what it received with what it transmitted. If the two agree, no collision occurred. Because other BIUs will hear the transmission (and therefore not attempt to transmit themselves) after a time which is at most the propagation delay of the cable, only the first part of a transmitted message needs to be checked in this manner. In the TMS, comparison of the first two bytes gives sufficient protection.

If a collision occurred, both BIUs must stop transmitting and retransmit their messages. To avoid repeated collisions as BIUs keep retrying transmissions, each BIU waits a pseudo-random time before retransmitting.

4.2.2 Packet Acknowledgements (ACKs)

When a BIU ascertains that it has "correctly" received a packet, it sends an acknowledgement signal onto the network. The acknowledgement signal (ACK) is one byte (not a packet) containing the address of the receiving BIU. When a BIU completes transmitting a packet onto the network, it

begins listening for this ACK. If the ACK is not received within 100 microseconds, it retransmits the packet.

An ACK will not be received if:

1) the destination address is that of a powered-off or non-existent BIU,

2) the destination BIU detected that a parity error occurred in the transmission of the message, or

3) the destination BIU temporarily had no buffer storage to accept the message.

In the first two cases, the originating BIU should receive no return transmission. In the third case, the destination BIU transmits a byte containing FF_{16} instead of the normal ACK. In all three cases the sending BIU attempts to retransmit the packet.

If a successful transmission is not achieved after 127 retries, the packet is considered non-transmittable and discarded by the transmitting BIU. Furthermore, if the transmitting BIU is the one attached to the MODCOMP, any immediately succeeding messages queued for the non-responding destination will be discarded. That is, once the MODCOMP BIU discards a packet, it will discard all to-be-transmitted packets until one is found which is meant for an alternate destination. At this point, the BIU "forgets" that the first destination was non-responding and will no longer discard packets to be sent to it. The purpose of this procedure is to prevent the MODCOMP BIU from tying up the network by attempting to send either a long multi-packet message of which one packet will be missing or a long multi-packet message to a powered-off or non-existent BIU.

In the first two cases, retransmissions are tried as soon as possible within the normal scheme of processing in the originating BIU. In the third case, however, a retransmission is not attempted for at least 256 milliseconds. The purpose of the delay is to allow the destination BIU time to transfer data to its subscriber and thereby to free up buffer space.

4.2.3 Parity Signals

Parity processing is done in the BIU at both the byte and packet level. At the byte level, parity processing is automatically handled by the BIU hardware (specifically the ACIA). The ACIA automatically calculates and transmits a parity bit for each byte it sends onto the net. The receiving ACIA automatically determines whether a parity error has occurred in the transmission of the byte. A flag, monitored by the BIU software, is raised if an error is detected.

At the packet level, parity processing is a software function. All bytes in a packet are "exclusive or-ed" together as they are being sent onto the network. After all message bytes have been transmitted, the resulting "exclusive-or" parity byte is sent onto the network. The destination BIUs software "exclusive-ors" together all bytes of a message as they are received. The destination BIU then reads the parity byte sent by the originating BIU. An error has occurred if the calculated and received parity types do not agree.

If either type of parity error occurs (byte or packet), the receiving BIU ignores the message and does not send an ACK. The originating BIU then resends the message.

4.3 Performance Monitoring

Three activities are required to do performance monitoring in the TMS bus system. Two of these are performed in the BIUs.

First, each BIU accumulates performance data by counting when significant performance-related events occur. Counts of eight events are accumulated - good transmissions, missing ACKs, collisions, discarded packets, good receptions, receptions with bad parity, receptions when no buffer space was available, and network busy when transmission was desired. The status counts are described more fully in Section 6.2.

Second, each BIU periodically reports its accumulated values. Every 60 seconds each BIU generates a status message containing these counters. The message is transmitted onto the net to a special network address (address 00).

Third, the routine reports must be processed. In the TMS, cumulative automatic processing is accomplished by software in the MODCOMP (see [3]).

PART II
BIU SOFTWARE MAINTENANCE NOTEBOOK

The following sections deal with loosely connected aspects of BIU programming. The sections are intended to supplement detailed documentation found in actual source program listings.

SECTION V
6502 ASSEMBLY LANGUAGE

5.0 INTRODUCTION

The assembly language used to program the 6502 is similar to other microprocessor assembly languages and is in general fairly simple and straightforward. The language does however have several characteristics which might initially confuse a novice 6502 programmer. Those characteristics which most confused the author are listed below.

The section is not intended to be a complete description of the 6502's assembly language. It is rather, assumed that the reader has at least a list of the 6502's instruction set, is familiar with basic assembler language programming, but is not cognizant of some of the restrictions or side effects of some of the 6502's instructions. The section is intended to shed light on what,at first glance, may seem to be unusual instruction sequences in the BIU's software. For details on each of the 6502's instructions see [9].

The notation adopted in this and following chapters is the following:

AND,OR,RTS,JMP, etc.	=	6502 instructions
"X", "Y", "A"	=	the 6502's X and Y index registers and accumulator
NAME	=	a subroutine in the software
"NAME"	=	a variable or statement label in the software
$9999	=	9999_{16} ($ is the hexadecimal symbol used in 6502 assembly language)

PAGE 28 INTENTIONALLY BLANK

5.1 Non-Interchangeability of Registers or Addressing Modes

The "X", "Y", and "A" registers cannot be used interchangeably. For instance, logical instructions (AND, OR, EOR) can only be done with the A register. Even the "X" and "Y" registers cannot be used interchangeably. "Indexed Indirect" addressing can only be accomplished with the "X" register; "Indirect Indexed" can only be done with "Y." Furthermore, not all addressing modes are available with each instruction. The available combinations are indicated in [7]. The effect of these restrictions is that, occasionally, register-to-register transfers or register-to-memory "saves" are required to set up an operation. Figure 5.1-1 shows shows an example of such a situation.

```
LDA     (TEMPTR),Y      Get value of interest
TAX                     Temporarily save it in X
LDY       #07           Get offset of destination (Part I)
LDA     (BIUPTR),Y      Get offset of destination (Part II)
TAY                     Put offset in required register
TXA                     Get value of interest back
STA     (BIUPTR),Y      Finally save value of interest
```

(adapted from the "SPMSGI" routine of the NOVA version of the BIU software)

Figure 5.1-1. Example of Data Transfers Required by the Non-Interchangeability of Registers

5.2 Conditional Branching - Comparing Two Numbers

At first glance there appear to be no 6502 instructions for traditional tests such as "Branch if greater than" or "Branch if less than or equal to." However, the CMP, CPX, and CPY instructions do set the status register bits so that such tests can be accomplished. When these instructions are used to compare two unsigned* bytes (one in a register - either "A", "X", or "Y" - and one in memory) the following tests can be performed on the status bits:

Branch if register < memory	Branch if Carry Clear	BCC
Branch if register ≥ memory	Branch if Carry Set	BCS
Branch if register > memory	Branch if Minus	BMI
Branch if register ≤ memory	Branch if Plus	BPL
Branch if register ≠ memory	Branch if not Equal	BNE
Branch if register = memory	Branch if Equal	BEQ

The compare instructions internally subtract the contents of the register from the contents of memory and set the status bits depending on the result, but leave the contents of the register and the memory location unchanged.

5.3 Conditional Branching - Testing Individual Bits

There are three ways of testing individual bits in a byte. First, the usual logical operations (AND, OR, EOR) are available for use with a bit mask for testing any bit or combination of bits. The byte being tested and the mask (one in register "A" and the other in memory) are operated on, the result is stored in "A", and the status register is set. The testing is then accomplished with the conditional branch instructions BNE or BEQ.

There is a second, more limited, way of testing individual bits. This second way does not require a mask. When a BIT instruction is executed, the two high order bits of the referenced byte are copied into the sign ("N") and overflow ("V") bits of the status register. Then the BPL (BMI) and BVC (BVS) instructions can be used to test if bit 7 is 0 (1) and bit 6 is 0 (1).

*"Unsigned byte" is a byte interpreted as having a value between 0 and 255. In the BIU code, all counters are considered to be unsigned bytes.

The third way of testing individual bits uses the BIT instruction with a mask. In addition to moving bits 7 and 6 of the referenced byte into the status registers, BIT also performs a logical "and" of the referenced byte with the contents of the "A" register. This will affect the "Z" bit in the status register the same as if an AND had been executed. The contents of the accumulator will not, however, be affected. Thus the accumulator need not be reloaded to be compared to a second mask.

The ability of the BIT instructions to test two bits without a mask and to use a mask in testing without affecting the "A" register make it extremely useful when several successive tests are to be performed on a byte. Figure 5.3-1 shows an example of the efficiency of the BIT instruction versus the AND instruction. Both sections of code will test the three high order bits of "FLAGS" in sequence and branch accordingly. To make the most efficient use of the "BIT" instruction, the flags tested most often should be positioned in the two most significant bits of a byte.

5.4 Addition and Subtraction

All operations in the 6502 are one-byte operations. Two-or three-byte operations can however be implemented using a series of one-byte operations and tests of the condition codes.

For example, the first two sets of code in Figure 5.4-1 increment a two-byte field in memory. The first set of instructions adds one to the quantity in memory, the second set adds the one-byte quantity in register "A" to the quantity in memory. In both cases the quantities are unsigned.* In both cases the least significant byte is operated on first. If there is a carry from this operation, the most significant byte is adjusted. Note that the INC instruction does not use or set the carry bit whereas the ADC instruction both uses and changes it. Thus, the carry bit must be cleared before the first one-byte add. In subtraction, the carry bit is interpreted as a "borrow" bit. Use of the SBC subtract instruction is similar to use of the ADC instruction, as indicated in the third set of code in Figure 5.4-1.

*Unsigned one-byte quantities are between 0 and 255, unsigned two-byte quantities have values between 0 and 65535.

Using the BIT Instruction

```
        LDA     FLAGS     Get byte containing bits of interest
        BIT     TEST3     Set up status register
        BMI     BIT8      Go to "BIT8" if high-order bit is set
        BVS     BIT4      " " "BIT4" if second bit is set
        BNE     BIT2      " " "BIT2" if third bit set
         }
TEST3   .BYTE   $20
```

Using the AND Instruction

```
        LDA     FLAGS     Get byte containing bits of interest
        BMI     BIT8      Go to "BIT8" if high-order bit is set
        AND     #$40      Test second bit
        BNE     BIT4      Go to "BIT4" if it is set
        LDA     FLAGS     Must get flags back since have
                            changed accumulator
        AND     #$20      Try third bit
        BNE     BIT2      Go to "BIT2" if it is set
```

Figure 5.3-1 Two Examples of Testing Individual Bits

Incrementing a Two-Byte Field

```
        INC     1,X     Add 1 to low-order byte (X points
                          to high byte)
        BNE     NOCARY  Should there be a "carry" to the
                          next byte?
        INC     0,X     Yes, add 1 to high-order byte
NOCARY
```

Adding to a Two-Byte Field

```
        LDA     1,X     Get low-order byte (X points to
                          high byte)
        CLC             Set up for add.. Don't carry into
                          this byte
        ADC     INCMNT  Add contents of INCMNT
        STA     1,X     Save new low-order byte
        LDA     0,X     Get high-order byte
        ADC     #00     Add 1 to it iff there was a carry
        STA     0,X     Save (new) high-order byte
```

Subtracting from a Two-Byte Field

```
        LDA     1,X     Get low-order byte (X points to
                          high byte)
        SEC             Set up for subtract... Haven't
                          borrowed from this byte
        SBC     DECMNT  Subtract contents of DECMNT
        STA     1,X     Save new low-order byte
        LDA     0,X     Get high-order byte
        SBC     #00     Subtract 1 iff there was a borrow
        STA     0,X     Save (new) high-order byte
```

Figure 5.4-1 Examples of Addition and Subtraction Using a Two-Byte Operand (Adapted from BIU Software)

5.5 Interrupt Handling

In general, three types of interrupt signals are distinguishable by a 6502. An executing program is able (through the SEI instruction) to inform the processor to ignore temporarily (until a CLI instruction is executed) two of the interrupt types. The third type (referred to as a Non-Maskable Interrupt) will always be processed.

The executing program is expected to provide routines for handling each of the interrupt types. The program is expected to have the addresses of these three routines stored in the last six bytes of addressable memory. The two bytes at location $FFFA and $FFFB must contain the address of the routine for handling non-maskable interrupts. The addresses of the other two interrupt handlers follow. The meanings of the three interrupt signals are defined by the connections of the corresponding signal lines of the 6502. The interpretation of interrupts in the BIU is discussed in Section IX.

5.6 Stack Processing

A "stack", consisting of a section of RAM and a Stack Pointer register, is available for program use. It is implicitly maintained through the use of subroutine-related instructions such as JSR, RTS, and RTI. It can also be explicitly maintained through the PHA, PLA, PHP, PLP, and TXS instructions.

The only use of the stack in the BIU software is for temporary storage of various registers during execution of a subroutine or during interrupt processing. The register contents are "pushed" onto the stack before the subroutine call or at the beginning of the subroutine. The contents are "pulled" from the stack at the end of the subroutine and restored into the registers. Care must be taken to ensure that all required registers are saved before they are changed in the subroutine. The JSR instruction and the interrupt processor save only a minimum of information. Any additional required information must be explicitly saved with the PHA and PHP instructions.

The stack resides in consecutive locations in RAM in the $0100 to $01FF address range. The stack pointer always points to the location which will be used for the next addition to the stack. The starting point of the stack is under program control through the TXS instruction. When the stack is known to be empty, e.g., at the beginning of a "reset" or initialization routine, a value loaded into the stack pointer register will define the location that is the beginning of the stack. Subsequent additions to the stack will be stored in consecutive descending locations. That is, the stack pointer, is decremented after each "push" onto the stack and incremented after each "pull" from the stack.

The one constraint in defining the location of the stack is that the high-order byte of the stack pointer is hardwired to be $01. The TXS instruction, and all other stack manipulation instructions, only affect the low order byte of the stack pointer.

In the BIU software, it is assumed that the stack will never contain more than 128 entries. Thus the stack pointer is initialized to $017F and the locations $0180 to $01FF are made available for other uses. It should be noted, however, that if, through error, more than 128 entries are put into the stack, the stack will "wrap around" in page one. That is, the 129th entry will be put in location $01FF, the 130th in $01FE, etc., thus destroying the contents of any variables allocated to these locations.

SECTION VI
DESCRIPTION OF ROUTINES

6.0 INTRODUCTION

The following pages describe each of the routines used in the MODCOMP and NOVA versions of the BIU software. As noted earlier, a number of routines are included in both versions, though in slightly different forms. The descriptions of such routines note any differences between their two versions.

The descriptions are not intended to define the routines completely. The general algorithms used to perform certain BIU functions have already been presented in Section IV. These descriptions are rather intended as an aid to understanding the overall structure of the BIU software and the parts individual routines play in that structure. The descriptions are intended to be used in program maintenance as a supplement to the actual code listing.

Table 6.0-I summarizes some of the information of this chapter. It lists the 29 routines used in either the MODCOMP or NOVA BIU software, summarizes the function of each routine, indicates how each routine is invoked, and indicates in which versions each routine is included, together with the degree of differences in the routine between versions. The routines are described in the order in which they are listed in the table. This corresponds to their order in the source code, except for version-specific routines.

6.1 RESET

Invoked by: A Reset interrupt.

When:

1. BIU is first turned on,
2. RESET button on BIU is hit, or
3. a RESET is triggered by software in the NOVA [NOVA version only].

Table 6.0-I BIU Software Subroutines: Function, Method of Invocation, and Degree of Difference Between Versions

ROUTINE	FUNCTION	ROUTINE INVOKED BY	VERSIONS INCLUDED IN	DEGREE OF DIFFERENCE BETWEEN VERSIONS*
RESET	Resets BIU's interfaces, internal variables	Interrupt	N,M	1
MLOOP	Main loop: calls processing routines	RESET	N,M	1
EXPTIM	Interrupts MODCOMP when "TIMER" expires	Various	M	
PUTM	Sends queued packets to MODCOMP	MLOOP	M	
GETM	Gets long message from MODCOMP	MLOOP	M	
WRITEM	Sends one packet to MODCOMP	PUTM	M	
READM	Gets one packet from MODCOMP	GETM	M	
REVINT	Reverses direction of interface	WRITEM	M	
NOVA	Handles I/O to NOVA through subroutines	MLOOP	N	
INTNOV	Interrupts NOVA	Various	N	
DOIN	Gets and queues one packet from NOVA	NOVA	N	
DOOUT	Sends queued packets to NOVA	NOVA	N	
PAKPT	Sets up packet to be sent to NOVA	Various	N	
SPMSGO	Processes packets queued from NOVA	MLOOP	N	
SPMSGI	Processes packets queued from network	MLOOP	N	
PUTSTR	Puts character string in packet to go to NOVA	Various	N	
NET	Sends queued packet to network	MLOOP	N,M	1
TIMOUT	Maintains time-of-day clock	MLOOP	N,M	2
CKTOUT	Checks whether software timers have expired	MLOOP	N,M	1
INTBUF	Sets up a buffer to hold input from net.	Various	N,M	2
IRQ	Handles interrupts from net and hardware timer	Interrupt	N,M	1
NMI	Handles NMI interrupts, turns BIU receiver on	Interrupt	N,M	3
PCONST	Allocates packet buffer and builds header	Various	N,M	2
STIMER	Sets three-byte timer	Various	N,M	3
CTIMER	Compares three-byte timer to "TOD"	CKTOUT	N,M	3
SFINC	Increments two-byte counter	Various	N,M	3
ALLOC	Allocates packet buffer	Various	N,M	3
ENQ	Puts buffer on specified queue	Various	N,M	3
DQ	Removes buffer from specified queue	Various	N,M	2

*1 – Differences in several consecutive lines in several parts of the routine
2 – Minor differences in several lines (e.g., different variable names or different values of constant)
3 – Absolutely identical

Parameters: None.

Actions:

1. Reinitialize control registers for interface devices (VIA, ACIA),
2. reinitialize buffer management variables; indicate that no messages are contained in the BIU's buffers,
3. terminate any existing link to another BIU when a signoff [NOVA version only],
4. Perform various other initializations.

Exit: Fall through and execute MLOOP.

Differences between NOVA and MODCOMP versions:

1. NOVA version allows for a partial initialization. Execution will pass to the statement labeled "RESTRT" when NOVA changes from a booted to an unbooted state (i.e., when the reset switch on the NOVA panel is hit).
2. NOVA performs different activities depending on whether or not the NOVA has been booted and whether or not the BIU has been previously signed on to another BIU.
3. Interface device control registers are set differently (see section 10.2.4).

6.2 MLOOP

Invoked by: Falling through from RESET.

When: At conclusion of initialization.

Parameters: None.

Actions: Loops continuously calling specialized subroutines to send data to the subscriber device, receive data from the subscriber device, send data onto the net, update clocks, allocate buffers for input from the subscriber device, and send status messages.

Exits:

1. By processing a Reset interrupt, or
2. By detecting that the NOVA has gone from the "booted" to "unbooted" state and invoking partial reinitialization [NOVA version only].

Differences between NOVA and MODCOMP versions:

1. Different routines are called to handle I/O to the subscriber device.
2. Only the NOVA version monitors the "NBOOT" flag.

6.3 EXPTIM

Invoked by: Various routines, using either the JSR instruction (in CKTOUT) or the JMP instruction (at all other times).

When: The variable "TIMER" is found to be 0; i.e., when an I/O operation to the MODCOMP has taken too long.

Parameters: None.

Actions:

1. Set "TO" bit to indicate a timeout has occurred and interrupt the MODCOMP, and
2. reset "TIMER" to 0.10 seconds.

Exit: Via an RTS instruction. Thus when called from CKTOUT with a JSR instruction, execution will return to CKTOUT. When called from other routines with a JMP instruction, control will pass to that routine's calling program which will in all cases be MLOOP.

Differences between NOVA and MODCOMP versions: Exists in MODCOMP version only.

6.4 PUTM

Invoked by: JSR from MLOOP.

When: Each cycle through MLOOP.

Parameters: None.

Actions: Sends any packets in the "QIN" queue into the MODCOMP. If there are any packets in the queue, a loop is executed which

1. interrupts the MODCOMP with the "INRDY" bit set,
2. waits for a read from the MODCOMP,
3. reverses the interface direction,
4. sends one packet to the MODCOMP via the WRITEM routine,
5. resets the interface direction.

The loop is terminated if the MODCOMP does not issue a read in response to the interrupt or if all packets have been sent. When all packets in the queue have been sent, the MODCOMP is interrupted with the "INRDY" bit cleared.

Exits: Via RTS if there are no packets in "QIN" queue, if MODCOMP can't be interrupted, if MODCOMP issues a write, or when QIN has been emptied. Via JMP to EXPTIM (and hence to MLOOP) if MODCOMP does not issue a read within 0.02 seconds of an interrupt, or if WRITEM detects a problem in the transfer.

Differences between NOVA and MODCOMP versions: Exists in MODCOMP version only.

6.5 GETM

Invoked by: JSR from MLOOP.

When: Each cycle through MLOOP.

Parameters: None.

Actions: Receives one "MODCOMP-sized" packet from MODCOMP and segments it into one or more "network-sized" packets which are placed in the "QOUT" queue. Includes provisions for handling retransmissions of packets from MODCOMP which were only partially transmitted earlier. Process includes:

1. reading the header of a packet being transmitted for the first time, or reading and discarding characters received earlier when a packet is being retransmitted;
2. looping through calls to READM getting a full network-sized packet from the MODCOMP each time; and
3. calling READM one last time to get partial network-sized packet to complete transmission from MODCOMP.

Exits: Via RTS when MODCOMP terminates its write, or via JMP to EXPTIM (and hence to MLOOP) if the MODCOMP takes too long to do the write.

Differences between NOVA and MODCOMP versions: Exists in MODCOMP version only.

6.6 WRITEM

Invoked by: JMP from PUTM.

When: MODCOMP has issued a read to the BIU.

Parameters: None.

Actions: Word-by-word transfer of the first packet in the "QIN" queue into the MODCOMP through the VIA. After some initialization, a loop is executed which transfers one 16-byte word of data at a time.

Exits: Normal exit is a JMP back into PUTM upon successful transmission of a packet. Two types of abnormal exit:

1. the transfer has taken longer than expected; thus the interface is reversed and control is returned to MLOOP via a JMP to EXPTIM.
2. the MODCOMP issued a write during the transfer; thus control is returned to MLOOP via a JMP to REVINT.

Differences between NOVA and MODCOMP versions: Exists in MODCOMP version only.

6.7 READM

Invoked by: JSR from GETM.

When: GETM expects data from MODCOMP.

Parameters: "C" is the index of the last word (16 bits) in the packet (pointed to by "TEMPTR") used by GETM; "CEND" is the index of the last word (16 bits) to be filled by "READM."

Actions: Word-by-word transfer of data from the MODCOMP through VIA into a packet designated by GETM. After some initialization, a loop is executed which transfers one 16-byte word of data at a time.

Exits: Via RTS to GETM. "CERROR" is set to 0 if normal exit, or is set to 1 if the transfer took too long, or is set to -1 if the MODCOMP terminates its write.

Differences between NOVA and MODCOMP versions: Exists in MODCOMP version only.

6.8 REVINT

Invoked by: JSR or JMP from WRITEM.

When: WRITEM has detected an error in writing to the MODCOMP.

Parameters: None.

Actions: Changes direction of interface from "set for output to MODCOMP" to "set for input from MODCOMP."

Exits: Via RTS. This will return control to WRITEM or to MLOOP depnding on whether a JMP or JSR is used in entry from WRITEM.

6.9 NOVA

Invoked by: JSR from MLOOP.

When: Each cycle through MLOOP.

Parameters: None

Actions: Handles all I/O with the NOVA through the use of the subroutines DOIN and DOOUT. If the NOVA has a write outstanding and a buffer is ready to accept data, DOIN is called to accept the data. If the NOVA has a read outstanding and a buffer is ready for it, DOOUT is called to write as much data as the NOVA will take. If no I/O is indicated, a check will be made to see if new buffers can be made ready for input or output. If so, the NOVA is interrupted (in INTNOV) with this changed status, and may respond with an I/O request.

Exits: Via RTS unless have to interrupt NOVA or call DOIN or DOOUT. DOIN and DOOUT are JMPed into and will do an RTS to MLOOP without returning to NOVA. INTNOV is fallen into or JMPed into and will also do an RTS to MLOOP.

Differences between NOVA and MODCOMP versions: Exists in NOVA version only.

6.10 INTNOV

Invoked by: JMP from NOVA, DOIN, or DOOUT; or falling through NOVA.

When: An interrupt to the NOVA is needed.

Parameters: The "A" register should contain the value of "PORT2B," the status signal to be sent to the NOVA.

Actions: The routine waits until the NOVA has no outstanding interrupts ("NDONE" = 1) and then interrupts it. It is assumed that the calling routine has verified that the NOVA has been booted.

Exits: RTS, which will pass control to MLOOP at the point immediately after the call to NOVA.

Differences between NOVA and MODCOMP versions: Exists in NOVA version only.

6.11 DOIN

Invoked by: JMP from NOVA.

When: NOVA has issued a write to the BIU ("INFLAG" = 1) and BIU has a buffer ready for data.

Parameters: None.

Actions: Performs word-by-word transfer of one packet from the NOVA through the VIA. Will take at most only one packet of data from the NOVA. (In the unusual case where handshaking with the NOVA somehow goes bad, no packets will be taken.) Only data is taken from the NOVA.

Packet headers are constructed by the PCONST routine which allocates a buffer for input from the NOVA. Once a packet is complete, it is put on the "QPROCO" queue for later processing by SPMSGO.

Exits: Two reasons to exit from this routine -

1. NOVA stops sending data ("OUTFLG" goes to 0), or
2. A packet has been filled up.

Four ways to exit from this routine -

1. Enqueue packet for later processing by SPMSGO, allocate a new packet for use by DOIN next time thru. Indicate still ready for traffic from NOVA.
2. Enqueue packet, but find no more packets available and NOVA still wants to talk. Indicate no longer ready for traffic from NOVA and interrupt NOVA.
3. Enqueue packet, find no more available but NOVA is finished talking. Indicate no longer ready for traffic from NOVA but do not need to interrupt it.
4. If somehow generated an empty packet (header data only), leave all flags and pointers as they were and do not enqueue the packet. All four cases will be terminated by an RTS to the MLOOP code, either directly or indirectly via an intermediate JMP to INTNOV.

Differences between NOVA and MODCOMP versions: Exists in NOVA version only.

6.12 DOOUT

Invoked by: JMP from NOVA.

When: NOVA has issued a read to the BIU ("OUTFLG" = 1) and "NBUSY" = 0) and BIU has at least one packet ready to be sent.

Parameters: None.

Actions: Performs word-by-word transfer of packets to the BIU through the VIA. DOOUT is called to do the output from the BIU to the NOVA. As many packets as possible are transferred. Partial packets may be transferred (with the remainder of the packet being transferred the next time through this routine). Packet headers are not sent to the NOVA. Packets to be transferred are taken from the "QIN" queue constructed by the SPMSGI and PUTSTR routines.

Exits: Two reasons to exit from this routine -

1. NOVA stops asking for data ("NBUSY" goes to 1 or "OUTFLG" goes to 0),
2. BIU has no more packets for NOVA ("QIN" goes to -1).

Two ways to exit from this routine -

1. If BIU still has data left for NOVA, save pointer to packet ("OUTPTR"), position in packet ("OUTBC"), and length of packet ("OUTPL").
2. If no more data, set flags read by both the BIU ("OUTSET") and the NOVA ("INRDY") to show this.

In either case, reverse the direction of the NOVA/BIU interface back to input, interrupt the NOVA, and return (via "INTNOV") to the MLOOP code.

Differences between NOVA and MODCOMP versions: Exists in NOVA version only.

6.13 PAKPT

Invoked by: JSR from NOVA or DOOUT.

When: A new packet is to be made ready for transmission to NOVA.

Parameters: The "X" register should contain the number of the buffer (0-20) containing the packet.

Actions: Sets up pointer and indexes ("OUTPTR," "OUTSET," and "OUTBC," "OUTPL") for packet specified by register "X". If that packet is of odd length, the byte following the last character of the data in the buffer is set to a null character. This extra byte fills out the last two-byte word that will be sent to the NOVA.

Exits: RTS.

Differences between NOVA and MODCOMP versions: Exists in NOVA version only.

6.14 SPMSGO

Invoked by: JSR from MLOOP.

When: Each cycle through MLOOP.

Parameters: None

Actions: Handles messages from the NOVA. It will pass on the messages to the network, will handle them itself, or will discard them, depending on what the BIU is expecting from the NOVA. This routine reads packets from the "QPROCO" queue constructed by the DOIN routine. This routine sends packets out to the

1. Network via the "QOUT" queue (later processed and actually transmitted by NET),

2. NOVA via the "QIN" queue through calls to PUTSTR ("QIN" is later processed and actually transferred by DOOUT).

There are three parts to this routine. Which part is executed depends on the state of the BIU:

1. If "CONECT" = 1, the BIU is not waiting for a reply from the NOVA to any BIU-generated message.

2. If "CONECT" = 0, the BIU is waiting for the NOVA to say whether it will accept another terminal's linking to it,

3. If "CONECT" = -1, the BIU is waiting for the NOVA to say which system it wants to talk to.

In the first case, the routine simply switches the packet to the "QOUT" queue. In the other two cases, the routine checks whether the received packet is a suitable response and takes appropriate action.

Exits: RTS.

Differences between NOVA and MODCOMP versions: Exists in NOVA version only.

6.15 SPMSGI

Invoked by: JSR from MLOOP.

When: Each cycle through MLOOP.

Parameters: None

Actions: Process a packet from the network. Either transmits the message to the NOVA, handles the message itself, or discards it, depending on the message type and the state of the BIU and NOVA. This routine reads one packet from the "QPROCI" queue constructed by the NINT network interrupt handler. If the packet is to be forwarded on to the NOVA, it is simply switched to the "QIN" queue where it will be handled by DOUT. This routine may also generate messages to the NOVA via the PUTSTR routine.

For purposes of this routine, the BIU/NOVA state is defined by three variables:

1. "NBOOT,"
2. "CONECT," described in SPMSGO,
3. "WAIT," also set by SPMSGO and CKTOUT indicating if the BIU is waiting for a sign-on acknowledgement from another BIU (1) or not (0).

Exits: Via RTS.

6.16 PUTSTR

Invoked by: JSR from RESET, SPMSGO, SPMSGI, or CKTOUT.

When: A character string packet generated within the NOVA BIU is to be transmitted to the NOVA.

Parameters: The "A" register should contain the offset of the beginning of the string to be sent; the offset is calculated from the address of "ASCII." The string is terminated by a null (zero) byte.

Actions: The routine builds a packet in the same format as a network packet and enqueues the packet on the "QIN" queue for the NOVA.

Exits: Via RTS. If a buffer could not be allocated for the packet, the "Y" register contains -1. (This condition is not tested in the current BIU code, however; it is assumed by the callers of PUTSTR that the message was sent to the NOVA.

Differences between MODCOMP and NOVA versions: Exists in NOVA version only.

6.17 NET

Invoked by: JSR from MLOOP.

When: Each cycle through MLOOP.

Parameters: None

Actions: Outputs one packet onto the network through the ACIA if possible. Several step process including:

1. delay if an "FF" (full) ACK was received to last packet sent or a collision detected from the last transmission,
2. verify net is free and complete packet header,
3. loop sending one byte at a time to the ACIA,
4. if good ACK not received, indicate that packet should be retransmitted later, and
5. if good ACK received, free up the buffer.

Exits: Via RTS. Six reasons to exit -

1. no packet to be transmitted,
2. the net is already busy,
3. a collision is detected ("XMIT" is set to 0 by NMI during transmission of the packet),

4. an ACK received that is OK,

5. an ACK is received indicating that the message was not received,

6. no ACK is received.

In cases 2 and 3 the message is saved for later retransmission. In cases 5 and 6 the message is saved if the retransmission count has not been exceeded.

6.18 TIMOUT

Invoked by: JSR from MLOOP.

When: Each cycle through MLOOP.

Parameters: None.

Actions: TIMOUT is responsible for keeping the variable "TOD" current by monitoring a hardware timer in one of the BIU's VIAs. "TOD" indicates current time-of-day. TIMOUT polls the timer to see if it has timed out. If it has, TIMOUT decrements "TICK." If "TICK" reaches 0, "TOD" is incremented, "TICK" is reset, and the "TOFLAG" is reset (by reading "TIMRL"). The periodicity of the timer is under BIU control. It is set to approximately 1/100th of a second in RESET.

Exits: RTS.

Differences between NOVA and MODCOMP versions:

1. "TOD" is expressed in seconds in the MODCOMP and in quarter-seconds in NOVA version (i.e., the NOVA "TOD" is updated every 25 ticks, the MODCOMP "TOD" is updated every 100 ticks).

6.19 CKTOUT

Invoked by: JSR from MLOOP.

When: Each cycle through MLOOP.

Parameters: None.

Actions: CKTOUT checks for two types of timeouts and takes appropriate actions if they are found. The first type of timeout is version-specific, the second is common to both versions. The three conditions checked are:

1. Is it time yet to interrupt the MODCOMP to assure that it is OK? [MODCOMP version]
2. Has this BIU waited too long for a sign-on acknowledgement from another BIU? [NOVA version]
3. Is it time to send a status message?

Exits: RTS when both conditions have been checked and appropriately handled.

Differences between NOVA and MODCOMP versions:

1. First condition checked, as described above,
2. NOVA puts status message on "QOUT" queue to be sent onto network. MODCOMP puts status message on "QIN" queue to be sent into the MODCOMP. (Later, when the message has been copied into the MODCOMP by the "WRITEM" routine, the message is transferred to the "QOUT" queue and then sent to the network.)
3. The time till the next status message is expressed as 240 quarter-seconds in the NOVA and 60 seconds in the MODCOMP.

4. NOVA references variables "BIUPTR" and "CURBIU," MODCOMP references variables "TEMPTR" and "CURMOD."

6.20 INTBUF

Invoked by: JSR from MLOOP, RESET, IRQ.

When: A buffer has just been filled with data from the net (IRQ), and at each reset and each cycle through MLOOP.

Parameters: None.

Actions: Reserves a buffer for holding network input if no buffer is so reserved. Initializes the required pointers and counts ("INTSET," "INPTR")

Differences between NOVA and MODCOMP versions:

1. If a buffer is required, the NOVA version will set one up only if there will still be one buffer left in the free-buffer stack. The MODCOMP version will however set up the last free buffer if it is required. The purpose of the NOVA version's behavior is to try to keep a free buffer for input from the NOVA.

6.21 IRQ

Invoked by: An interrupt from

1. a timer running in a 6522 chip [MODCOMP version only]
2. the network ACIA.

When: Interrupts are unmasked and the timer TIMER 2 [MODCOMP only] expires, or interrupts are unmasked, the BIU's receiver is enabled, and traffic is sent on the net.

Parameters: None

Actions:

1. If the interrupt is from TIMER 2 and the software timer "TIMER" is not zero, decrement "TIMER" and reset and restart TIMER 2 [MODCOMP version only],

2. Otherwise, if the interrupt was caused by a transmission from this BIU, accumulate 2 bytes and compare them against the bytes transmitted. If the bytes match, turn off the receiver. If they do not match, turn off both the receiver and the transmitter, increment the status field counting collisions, and wait a pseudo random interval before retransmitting.

3. If the network ACIA interrupt was caused by the reception of a transmission from another BIU, check whether the first byte of the destination address matches the first byte of the BIU address. If not, turn off the receiver.

4. If the packet is for the BIU, loop until all characters of the packet have been read from the network (check parity for each character as it is read, and compute the "exclusive-or" packet parity byte). If there are no parity errors, send an acknowledgement after the packet is received, unless there is no buffer free for the packet. In that case, send a special ACK of $FF.

5. If the packet is successfully received, enqueue it for transmission to the subscriber. The queue "QPROCI" is used in the NOVA BIU and the queue "QIN" is employed in the MODCOMP.

Exists: Via an RTI instruction.

Differences between NOVA and MODCOMP versions:

1. The NOVA version has no TIMER 2.
2. Slightly different algorithms are used in testing whether a packet is for this BIU since the NOVA BIUs respond to both a data and a "boot" address, while the MODCOMP responds to a data address and the status message broadcast address.
3. The MODCOMP BIU handles sign-on requests (requests from other BIUs to establish a logical communications link) in this routine. The NOVA BIU does not make a check for such requests in this routine since the NOVA is consulted before a response is given to a sign-on request.

6.22 NMI

Invoked by: The turning off of a carrier on the network.

When: Any BIU which has been transmitting on the net ceases to transmit. This interrupt cannot be masked and will occur in all BIUs connected to the net (including the BIU which was transmitting).

Parameters: None.

Actions:

1. The "I-was-transmitting" flag "XMIT" is reset.
2. The BIU's receiver is turned on.

Exit: Via an RTI instruction.

Differences between NOVA and MODCOMP versions: None

6.23 PCONST

Invoked by: JSR from CKTOUT, GETM, RESET [NOVA version only], NOVA, DOIN, SPMSGO, CKSTAT.

When: A packet is needed.

Parameters: The "A" register should contain the type of message to be put in the packet (see Section 7.3 for possible codes).

Actions: Calls ALLOC to get a buffer, sets up pointers to it, and fills in essential information in the packet header:

1. Destination address, from "XADDR"
2. Origin address, value of "HOME"
3. Message type, from parameter
4. Packet length, initialized to 7 as a default.

Exits: Via RTS with "N" bit on and "Y" register set to $FF if no buffers available, or with "N" bit off and "Y" register set to 7 if buffer set up.

Differences between MODCOMP and NOVA versions: MODCOMP version references the variables "TEMPTR" and "CURMOD," where the NOVA version uses the names "BIUPTR" and "CURBIU."

6.24 STIMER

Invoked by: JSR from CKTOUT and SPMSGO.

When: A status message is sent out (CKTOUT), or a sign-on request message is sent out [NOVA version only].

Parameters: The "X" register should point to a three-byte variable which will be used as a deadline timer. The "A" register should contain a time increment.

Actions: The three-byte variable pointed to by the "X" register is set to the sum of the three-byte variable "TOD" plus the value of the "A" register.

Exits: RTS.

Difference between MODCOMP and NOVA versions: None.

6.25 CTIMER

Invoked by: JSR from CKTOUT.

When: Checking to see if it is time for a status message to be sent out, or if it is time to stop waiting for a response to a sign-on request.

Parameters: The "X" register should point to a three-byte timer variable.

Actions: CTIMER compares the specified timer variable to the current value of "TOD." If the timer is less than "TOD," the "CARRY" bit in the status register is cleared on exit, otherwise it is set.

Exits: RTS.

Differences between MODCOMP and NOVA versions: None.

6.26 SFINC

Invoked by: JSR from various routines.

When: An event occurs that is being monitored and recorded for reporting in a status message (e.g., a successful transmission has been concluded).

Parameters: The "X" register should point to the two-byte counter to be incremented.

Actions: Add one to the specified coutner.

Exits: RTS.

Differences between MODCOMP and NOVA versions: None.

6.27 ALLOC

Invoked by: JSR from various routines.

When: A buffer from the stack of free buffers is desired.

Parameters: None

Actions: If a free buffer is available, its index is stored in the "X" register, the "N" bit is cleared, and the free buffer stack is updated. If no free buffer is available, the "N" bit is set and the "Y" register is set to $FF.

Exits: RTS.

Differences between MODCOMP and NOVA versions: None.

6.28 ENQ

Invoked by: JSR from various routines.

When: A packet has been received, or created, and is ready to be put on a queue (either "QIN," "QOUT," "QPROCI," or "QPROCO").

Parameters: The "A" register should contain the offset of the queue header from the variable "NEXT" (e.g., LDA #QOUT-NEXT). The "Y" register should contain the index of the buffer to be added to the queue.

Actions: The packet is put at the end of the queue. The queue is linked together via pointers (indices) in the "NEXT" array. The last entry in the queue has its "NEXT" field set to $FF. The "A" register is set to $FF.

Exits: RTS.

Differences between MODCOMP and NOVA versions: None.

6.29 DQ

Invoked by: JSR from various routines.

When: A packet buffer is to be removed from a queue and added to the free-buffer stack because the message has been transmitted, processed, or ignored.

Parameters: The "X" register is the offset from "NEXT" of the entry in the queue immediately proceeding the one to be removed from the queue.

Actions: The buffer following that specified is linked to the one preceding it in the queue. The buffer specified is added to the free buffer stack.

Exits: RTS.

Differences between MODCOMP and NOVA versions: The two versions contain the exact same instructions, however the MODCOMP version includes a secondary entry point. The FREE entry point simply frees a buffer without removing it from a queue. It is used for handling buffers which were never placed in a queue.

SECTION VII
PACKET STRUCTURE

7.0 NETWORK PACKETS

Messages transmitted by BIUs on the network are broken into packets, which consists of an 8-byte header and (for data packets) up to 120 bytes of data. Packets consisting of only a header are sent from one BIU to another to accomplish various network control functions, such as the establishment of a logical link between BIUs. Figure 7.0-1 shows the format of the 8-byte header, together with a summary of values presently used in the header fields in the TMS. The following paragraphs contain additional information about some of the header fields; a more detailed description is found in [4].

7.1 Network Addresses

The packet header format allows space for two bytes of network address for both the originating BIU and the destination BIU. In the TMS system, however, only the first byte of the address is currently used; the second byte of each address is always set to zero.

The NOVA BIUs in the TMS respond to two different network addresses, one of which is the "data" address (for data messages to be sent to a program executing in the NOVA) and the other of which is the "boot" address (for code which is to be loaded into the NOVA for execution).

The MODCOMP BIU also responds to two addresses: its own data address and a special broadcast address used by all BIUs on the network as the destination for status messages. The status messages are passed to the MODCOMP where a special program accumulates the status counts for reporting on demand. The MODCOMP BIU acknowledges both data messages sent to it by other BIUs and status messages sent by other BIUs. Status messages from the MODCOMP BIU itself are sent directly into the MODCOMP and are also transmitted on the net. Since, however, a BIU does not accept and process its own messages, the MODCOMP BIU does not wait for an acknowledgement to the status messages it transmits on the network.

BYTE NO.	CONTENTS	VALID VALUES IN THE TMS	COMMENTS
0 (byte 1 is presently unused)	Destination Address	\$00 = Status Msg. Processor \$01 = Alternate TMS (MODCOMP) \$02 = TMS (MODCOMP) \$30–\$39 = Data addresses for Terminals 1–9 \$B0–\$B9 "Boot" addresses for Terminals 1–9	00, \$B0–\$B9 are valid only as destination \$B0–\$B9 are used by MODCOMP when Booting a terminal \$30–\$39 refer to a terminal which has been booted Other addresses can be used in extensions to the TMS
2 (byte 3 is presently unused)	Origin Address		
4	Sequence #	0–127	Used for quality assurance
5	Packet Type	\$02 – Data Packet \$DB – Status Msg. \$DE – Sign–off Notice \$DF – Sign–on ACK \$E2 – Sign–on Request	Other packet types can be used in extensions to the TMS
6	Retransmit #	0–127	Used for quality assurance
7	Packet Length	7–127	One less than total number of bytes (including header)

Figure 7.0–1 Packet Header Format

7.2 Quality Assurance Fields (Sequence Number and Retransmit Number)

Unique messages emitted from each BIU are sequentially numbered. Numbering starts at 0, runs to 127, and then wraps around to 0. A message may be repeatedly transmitted if no acknowledgement of reception is sensed. Successive retransmissions receive successive retransmit numbers from 0 to 127. After 127 retransmissions, the message is considered untransmittable and is discarded. The retransmission number is incremented when the receiving BIU fails to answer or responds that is has no buffer storage available. The retransmission number is not incremented for retransmission due to message collisions on the net.

The sequence number is used by the receiving BIU in its quality assurance processing. It is conceivable that the receiving BIU could get two copies of the same message. This would happen if its acknowledgement to the sender is somehow not properly transmitted and the sender thus retransmits the message. To prevent forwarding of duplicate messages on to the subscriber device, the sequence number of each newly-received message is compared to that of the last-received message. If the numbers are equal, the newly received message is ignored. An ACK is, however, sent in any case to avoid repeated retransmissions.

The retransmit field is not used at all by the receiving BIU.

7.3 Packet Type

Five types of packets are presently transmitted on the TMS network:

Data Packet ($02) - The packet contains data to be forwarded to the destination subscriber device. The packet requires no special processing by the BIU.

Status Message ($DB) - The packet contains status information to be forwarded to the status message processor. The contents of the data field of this packet type are detailed in the following paragraph. The destination field of a status message packet is the address of the status message processor, $00.

Sign-on Request ($E2) - The packet represents only a BIU-to-BIU message and contains no data. The sending BIU (which must be a terminal) is requesting that a "link" be established with the destination BIU. Sign-on requests are generated only by the NOVA BIUs and may be directed to any BIU on the system (except the LISTENER BIU described in [5]).

Sign-on Acknowledgement ($DF) - The packet is primarily a BIU-to-BIU communication in which the sending BIU is affirmatively responding to the destination BIUs request to establish a link.

Sign-on Notice ($DE) - The packet represents only a BIU-to-BIU message and contains no data. The sending BIU is notifying the destination BIU that it is unilaterally terminating their "link." The NOVA BIU software will generate a sign-off notice only when the RESET routine is executed. The MODCOMP BIU software will not generate a sign-off notice.

7.4 Status Messages

Status messages are generated by each BIU approximately every minute. The status message consists of a packet header plus 34 bytes of status information. The status information consists of

- Eight measures (counts) of the BIU's activities over the last minute,
- Six fields representing aspects of the BIU's current state at the time of message generation, and
- Twelve characters which by convention identify the type of BIU generating the message ("MEG" vs. "MOD") and the date (version) of its software.

Table 7.4-I shows the fields of a status message.

7.5 Subscriber Device Transfers

7.5.1 MODCOMP

The MODCOMP BIU exchanges data with the MODCOMP in packet format, much like the way it deals with the net. Packets received from the

Table 7.4-I Status Message Fields

	Field Number	Bytes	Description
HEADER	1	0,1	Destination address (always $0000)
	2	2,3	Origin address
	3	4	Sequence number
	4	5	Message type ($DB)
	5	6	Retry count
	6	7	Packet length ($29)
PERFORMANCE STATISTICS OVER THE LAST MINUTE	7	8,9	Number of packets successfully transmitted and ACKed
	8	10,11	Number of retransmissions because of no ACK reply to the packet
	9	12,13	Number of collisions
	10	14,15	Number of packets discarded after multiple retransmissions
	11	16,17	Number of packets received with a good longitudinal parity byte
	12	18,19	Number of packets received with a bad longitudinal parity byte
	13	20,21	Number of packets not accepted because of lack of buffer space to hold the packet
	14	22,23	Number of times a transmission was deferred because another BIU was transmitting
CURRENT STATUS	15	24	Number of packets queued for transmission (not including this status packet)
	16	25	Number of packets from the network queues for the subscriber
	17	26	Network ACIA status register (see Section X)
	18	27	Device ACIA status register
	19	28	VIA parallel port/timer interrupt register contents
	20	29	High order byte of last destination address referenced
BIU ID	21	30-41	ASCII identification of BIU

NOTE: Fields 7 - 14 will not wrap around to zero when they reach $FFFF; they are cleared after each status packet is sent.

net for the MODCOMP are forwarded on intact and including both the header and data. Likewise data received from the MODCOMP is in packet format (header plus data). Packets originating in the MODCOMP are, however, different from network packets in two ways:

1. the origin, sequence number and retransmit number fields are not set in MODCOMP-generated packets,
2. the maximum length of a MODCOMP-generated packet is 1024 bytes, while the maximum length of a network packet is 128 bytes, and
3. no trailing parity byte is included in a MODCOMP-generated packet.

The MODCOMP LIU transforms a MODCOMP-generated packet into one or more network packets by supplying the missing fields and segmenting it into as many packets as required (and attaching appropriate headers).

7.5.2 NOVA

The NOVA BIU interfaces with the NOVA not on a packet basis, but rather on a data-word-count basis. Headers are not sent to, or received from, the NOVA. A NOVA output-to-the-BIU operation may generate one or more network packets. A NOVA input-from-the-BIU may request only part of a packet or several packets.

SECTION VIII
MESSAGE BUFFER MANAGEMENT

8.0 INTRODUCTION

One of the functions of the BIU is to buffer messages intended to be sent to both its subscriber device and the network. Both the NOVA and MODCOMP BIU software versions provide 21 buffers for message packets. Each buffer is 128 bytes long.

Buffer management utility routines and procedures are required in both software versions to keep track of which buffers are empty, which contain data, and what is the next step in processing that data. All buffers are placed in a "free buffer" pool when the RESET routine is executed. They are than allocated and freed on a demand basis to hold messages to and from the subscriber. The buffer management procedure is essentially similar in the two versions. The NOVA version is slightly more complex as the BIU is required to do more processing of messages.

8.1 Message Flow

Buffered messages are held in first-in-first-out queues in both versions. Figure 8.1-1 indicates the queues maintained in each version and the primary routines which add and remove packets from each queue. In the NOVA version, all messages received from the net are added to the "QPROCI" queue. The SPMSGI routine then takes messages from this queue, does any processing required for the BIU, and transfers any packets intended for the NOVA to the "QIN" queue. In the MODCOMP, the process is compressed. The IRQ routine does what limited processing of messages is required of the BIU (i.e., processes sign-on packets) and puts messages for the subscriber in the "QIN" queue. A parallel situation exists for transmission of messages from the subscriber device to the net.

8.2 Queue Implementation

When a message is received by a BIU (or generated by a BIU) it is placed in a specific message buffer. Though the message may be logically

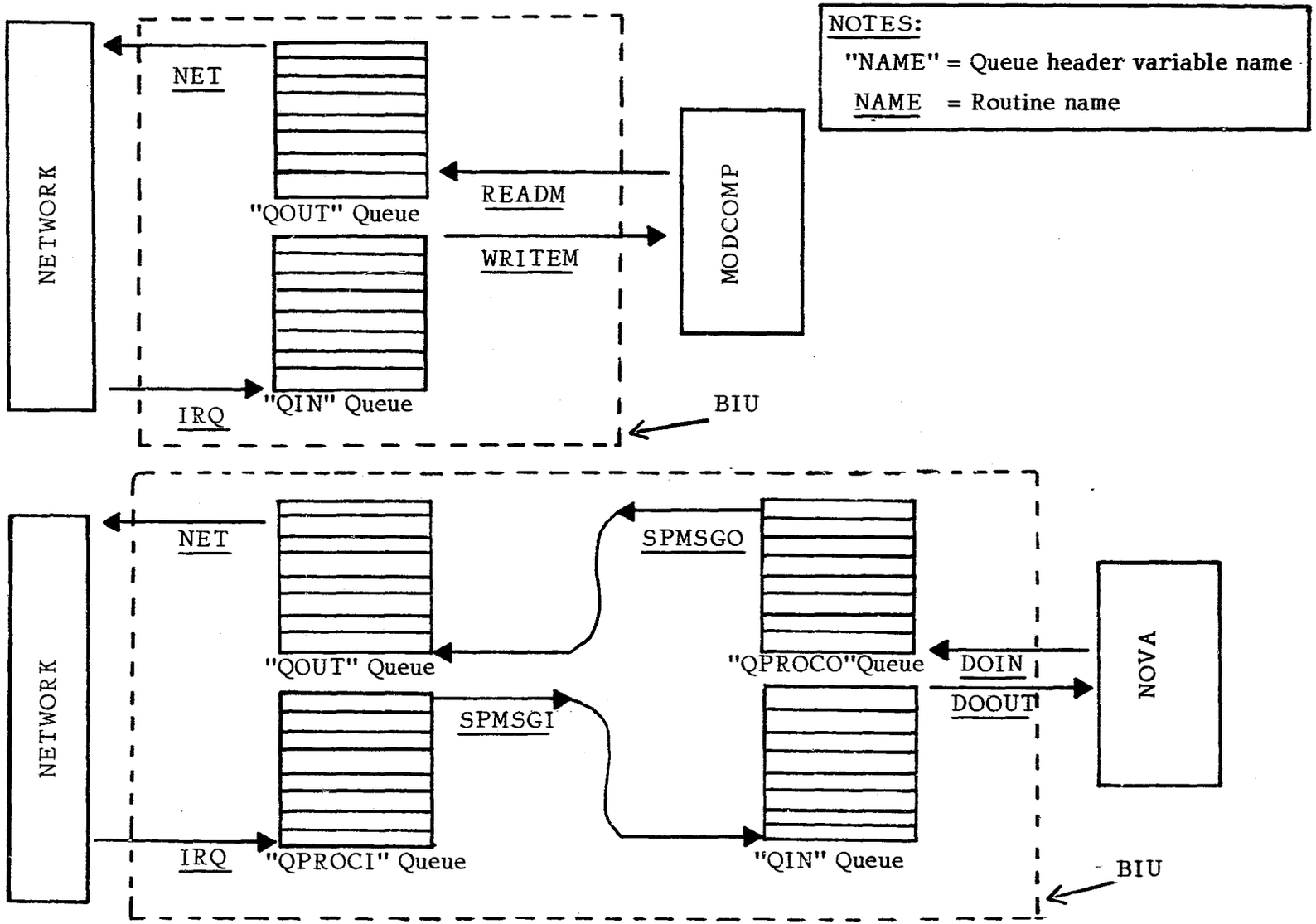


Figure 8.1-1 Buffer Queues and Queue Maintenance Routines for MODCOMP and NOVA BIUs

moved through several queues before its processing is finally completed, it is never physically moved. The logical movement is supported by assigning a number from 0 to 20 to each buffer and using a series of pointer variables:

1. "HIPTR" and "LOPTR" are 21-entry arrays containing respectively the high- and low-order bytes of the address of the beginning of each buffer,
2. "QIN", "QOUT", "QPROCI", and "QPROCO" are one-byte variables which contain the number of the buffer/message on the top of each queue (they will contain -1 if the queue is empty), and
3. "NEXT" is a 21-entry array containing the number of the buffer immediately following each buffer in a queue. (or -1 if the buffer is at the end of a queue).

"HIPTR" and "LOPTR" are set in the RESET routine and are never changed. They are referred to only to read or write a specific buffer.

The use of the queue-header variables and the "NEXT" array can best be explained by example. Figure 8.2-2 shows what values those variables would have if the contents of the queues were:

QIN - buffers $03, $05 and $04
QOUT - buffers $00, $01, and $02
QPROCI - buffers $06, $07, and $13
QPROCO - buffers $14 and $08.

A buffer is added to a queue by altering the entry in the "NEXT" array for the last buffer in the queue and the entry for the buffer to be added. Similarly, to delete a buffer from a queue, the "NEXT" value of the buffer "above" the one to be deleted must be changed. These functions are accomplished by the ENQ and DQ subroutines.

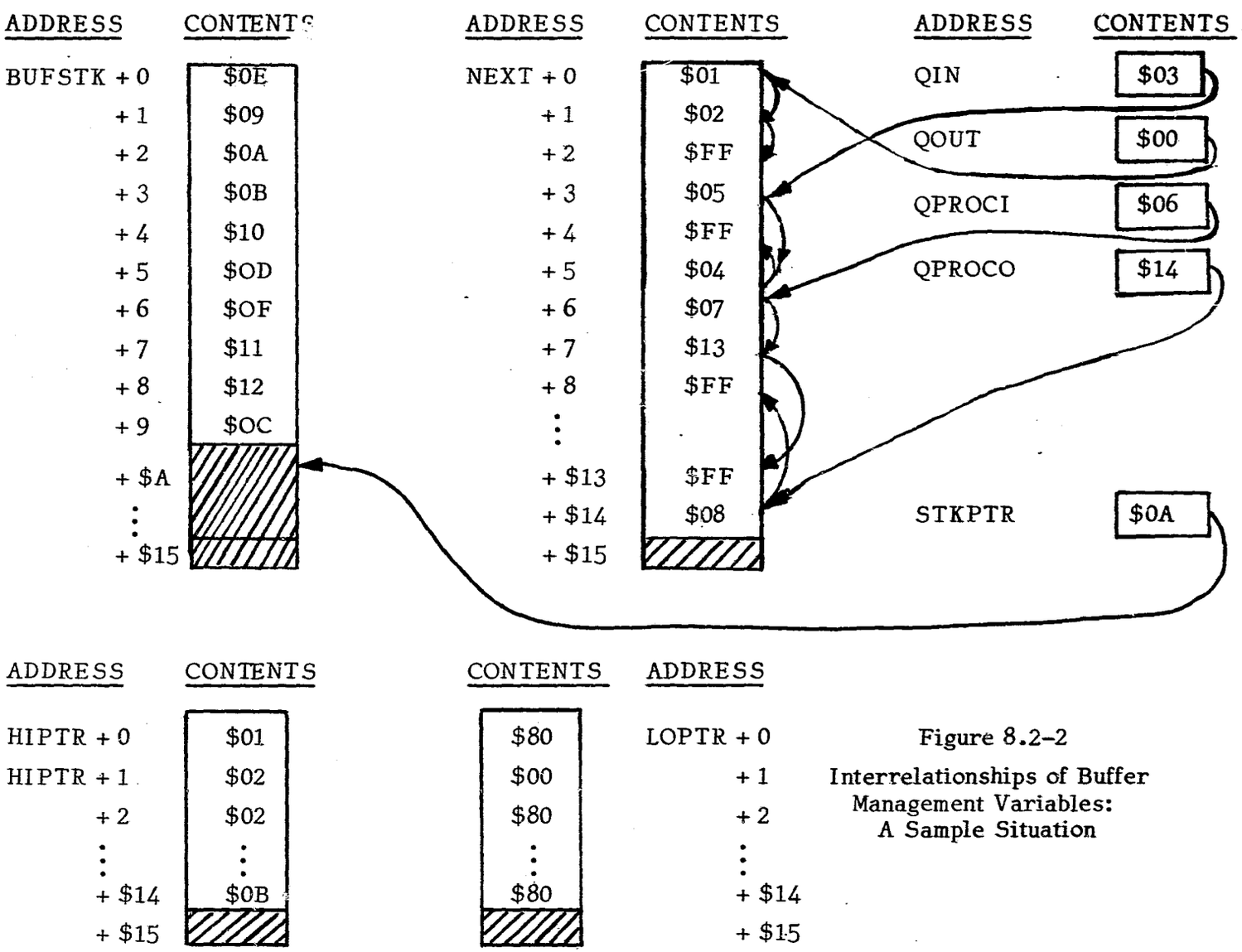


Figure 8.2-2

Interrelationships of Buffer Management Variables: A Sample Situation

8.3 Free-Buffer Management

A buffer is "free" if it does not contain a queued message and it is not reserved for a specific future use. The 21-entry array "BUFSTK" and the one-byte index "STKPTR" are used to maintain a stack of indices of buffers which are free. "STKPTR" contains the number of buffers which are free. If no buffers are free, STKPTR is set to -1. The indices of the free buffers are stored in consecutive locations starting at "BUFSTK". For example, Figure 8.2-2 indicates a situation where 10 buffers are free.

When a buffer is freed, it is "pushed" onto the top of the stack. That is, its index is stored in the next empty location of the "BUFSTK" array (which is pointed to by "STKPTR"), and "STKPTR" is incremented. A request for a free buffer is satisfied by "pulling" the top buffer from the stack. The ALLOC and DQ routines are used for all "push" and "pull" operations.

8.4 Reserved Buffers

The BIU software attempts to have empty buffers available at all times for certain specific purposes. Such reserved buffers are in neither the free buffer stack or the processing queues. The indices of these reserved buffers are stored in dedicated pointer variables. When a reserved buffer is used for its intended purpose. another buffer is removed from the free buffer stack and reserved.

In the MODCOMP version of the software a buffer is reserved for only one purpose - input from the network. The reserved buffer's address is stored in the two-byte variable "INTPTR" and its index is stored in "INTSET". "INTSET" is set to -1 if there is currently no buffer reserved. The INTBUF routine is responsible for attempting to maintain a reserved buffer so that messages from the net will not be lost.

The NOVA version of the software also attempts to reserve a buffer for network input using the same variables. Additionally, the NOVA version attempts to reserve a buffer for input from the NOVA. The variables "INSET" and "INPTR" are used for this purpose.

SECTION IX
INTERRUPT PROCESSING

9.0 INTRODUCTION

Interrupt processing is an important part of the BIU's interface protocols with both the network and the subscriber device. The BIU receives interrupts generated by network events, by the subscriber device, by the BIU's own internal timers, and by the RESET button on the BIU's faceplate. The BIU can, in turn, generate interrupts to the subscriber device. The following section considers these two aspects of interrupts - handling and generation. The discussions will make reference to software routines (NAME) and variables ("NAME") described in more detail elsewhere in this document (Sections VI and X).

9.1 Handling of Interrupts to the BIU

There are three independent hardware interrupt lines into the 6502. When a signal is received by the 6502 on any of these lines and the corresponding interrupt type is enabled, an interrupt will occur. There are three interrupt handling routines in the BIU's software corresponding to the three hardware interrupt lines. The three routines have names (RESET, IRQ, and NMI*) corresponding to the names generally used in descriptions of the hardware lines.

The 6502 determines which routine to execute by means of an "interrupt routine address vector". The 6502 assumes that the last six bytes of PROM storage ($FFFA - $FFFF) contain the addresses of three interrupt handling routines. The following paragraphs discuss why each interrupt type will occur, what the BIU software will do in response, and how and when each interrupt type is enabled.

*IRQ is an abbreviation for "Interrupt Request" and NMI is short for "Non-Maskable Interrupt".

9.1.1 RESET Interrupts

Occur When:

1. The Reset button on the front panel of the BIU is depressed, or
2. The routine RSET in the NOVA's bus handler is executed (NOVA version only, see [3]).

BIU's Response: Clear the processor stack, discard any messages held in the BIU's buffers, terminate any logical link with another BIU, reinitialize all variables used in the BIU's software, reinitialize interface protocol lines, and perform certain device specific initializations.

Exit: Fall through into the main loop of the code.

Enabled: At all times. This interrupt type cannot be disabled.

9.1.2 IRQ Interrupts

Occur When:

1. Any BIU on the network begins transmitting a message onto the network. Specifically, whenever bit 6 of the "NUARTS" status register in any BIU is changed from 1 to 0. Even the transmitting BIU receives an IRQ interrupt.
2. A hardware timer in the BIU (specifically, in the 6522 VIA parallel interface unit) expires [MODCOMP version only].

BIU's Response: Decide which of the two types of IRQ interrupts occurred by looking at "NUARTS" [MODCOMP version only]. If the interrupt is not from the network, it is assumed to be from the timer. In this case, the BIU simply decrements a software timer. If the message is coming across the net, the response depends on who is sending the message, who the message is for, and whether or not the message has been garbled by a collision. The status of the message can be determined by

looking at the first few bytes. Briefly, if the BIU detects that it is receiving its own transmission, it first decides if the transmission has been garbled. If it has, the BIU turns off both its transmitter and its receiver (this will cause an NMI interrupt). If the message appears to be intact, only the receiver is turned off. If the BIU detects that it is neither the originator or destination of the message, the receiver is turned off. If the BIU decides that it is the destination of the message, the message is received byte-by-byte and saved for later processing.

Exit: Via a RTI (Return from Interrupt) instruction. Return to the processing which was suspended when the interrupt occurred.

Enabled: Only when the "I" bit of the processor stastus register is clear. This bit is set whenever the SEI instruction is executed or whenever any of the three types of interrupts occur. The bit is cleared by the CLI instruction.

In general, IRQ interrupts are disallowed (masked) in the BIU when 1) changes are being made to any of the packet queues, 2) changes are being made to the free-buffer stack, or 3) when the BIU is awaiting an acknowledgement of receipt of a packet it has sent out.

In Cases 1 and 2, interrupts are disallowed because the interrupt handler might also try to update these structures. In Case 3 interrupts are disabled so that an acknowledgement is not confused with an incoming message. Acknowledgements are processed by the NET subroutine.

9.1.3 NMI Interrupts

Occur When: Any BIU on the network stops transmitting a message onto the network. Specifically, whenever bit 6 of "NUARTS" in any BIU is changed from 0 to 1. Even the BIU ceasing transmitting will receive an NMI interrupt.

BIU's Response:	Reset the BIU's receiver on and transmitter off by altering "NUARTS" and set "XMIT" to zero to indicate that such a reset has occurred.
Exit:	Via an RTI (Return from Interrupt) instruction. Return to the processing which was suspended when the interrupt occurred.
Enabled:	At all times. This interrupt type cannot be disabled.

9.2 Generation of Interrupts to the Subscriber Device

How

Interrupts are sent to the subscriber device (either the NOVA or the MODCOMP) by pulsing the interrupt line (the high-order bit in "PORT2B"). In the MODCOMP version the bit is labeled "EXTSIN"; in the NOVA version it is "NINTRP". In either case, the bit is normally 1. Clearing the bit to 0 will generate an interrupt in the subscriber device. The bit is then immediately reset by the BIU.

When

In the NOVA version, an interrupt should not be generated unless the NOVA is both booted ("NBOOT" = 0) and has no interrupts pending from the interface ("NDONE" = 1). If the NOVA is not booted, there is no handler to respond to the interrupt. If the NOVA already has an interrupt pending from its interface board (which will tell it that a transfer has been completed), a second interrupt will be ignored rather than queued.

The MODCOMP should not be interrupted unless it has an outstanding read to the BIU ("BUSYN" = 0). If an interrupt is sent to the MODCOMP while it is attempting to write to the BIU, the write will be garbled and irrecoverable. If an interrupt is sent to the MODCOMP while no I/O is pending, the I/O handler software in the MODCOMP will write an error message to the operator's console. Thus the TMS interface software in the MODCOMP attempts to have a "read" to the BIU outstanding at all times

(except when it is writing to the BIU) and thus be able to receive interrupts.

See also [3] for a further discussion of interrupts of both the NOVA and the MODCOMP.

Why

The primary reason for interrupting the subscriber device is to notify it of a change in the state of the BIU. When the subscriber device is interrupted it reads the BIU's status lines in "PORT2B". These lines are read only when the subscriber is interrupted.

The status lines are read by the interface hardware in the subscriber and then passed on through several layers of software to the interface software (see [3]). The interface software then can initiate or cancel I/O operations to the BIU, or take other appropriate actions, based on the new BIU status.

In the NOVA version, there is only one section of code (the INTNOV routine) for interrupting the NOVA. The code is executed under six circumstances:

1. When an error condition is detected by the routine NOVA (when the NOVA wants to write to the BIU but the BIU has no free buffer for input). A flag is set to indicate that an error is found; however the handler in the NOVA does not check the flag. This condition should not occur in normal operation.

2. When the routine NOVA detects that the NOVA interface is busy ("NBUSY" = 0) but it appears that an I/O operation has become hung ("INFLAG" = 0, "OUTFLG" = 0). The interrupt terminates the I/O operation and frees the "hung" condition.

3. When the routine NOVA detects that it can allocate a buffer for data from the NOVA when it could not do so earlier ("OUTRDY" changes from 0 to 1).

4. When the routine NOVA detects that it has data for the NOVA when it did not have data earlier ("INRDY" changes from 0 to 1).

5. When the routine DOIN detects that, during a transfer, it has no more buffers available for data from the NOVA, but the NOVA still is trying to write to the BIU ("INFLAG" = 0).

6. When the routine DOOUT terminates a write to the NOVA and reverses the interface, but the NOVA still has an outstanding I/O operation ("NBUSY" = 0).

In the MODCOMP version, interrupts are generated from three different places in the software:

1. The routine EXPTIM interrupts the MODCOMP when the "TIMER" expires and the "TO" bit is set. This happens when an I/O operation seems to be taking too long. The interrupt is generated to ascertain that the MODCOMP has not crashed or become hung. It is hoped that if the MODCOMP is hung, interrupting it may get it restarted correctly.

   "TIMER" settings and lengths of reads and writes have been chosen so that under normal conditions, I/O operations for transfers of data (not "dummy reads") should not be interrupts by a time out. While the BIU/MODCOMP interface is idle, however, the MODCOMP's outstanding "dummy reads" will be routinely interrupted by the BIU at regular intervals. The interrupts may thus be interpreted as the BIU saying "I'm okay; are you okay?"

2. The routine PUTM interrupts the MODCOMP each time it wishes to send it a packet. The MODCOMP should see the "INRDY" bit on, terminate any outstanding I/O, and respond with a read.

3. The routine PUTM also interrupts the MODCOMP when it has transmitted the last of its stored packet to the MODCOMP and turned the "INRDY" bit off. The MODCOMP should respond by terminating its read and initiating a dummy read or a write.

SECTION X
WORD-BY-WORD INTERFACE PROTOCOLS

10.0 INTRODUCTION

The "Message Buffer Management" section (Section VIII) described briefly the flow of message packets from the subsriber through the BIU and to the network (and vice versa). The focus of this section is a lower-level description of the byte-by-byte exchanges of data between the BIU and the subscriber and between the BIU and the network.

The network interface is considered first because it is the simpler of the two. Only one peripheral device (an ACIA) is required, and that device has only status, control, and data registers. Furthermore the network interface is the same for both the MODCOMP and NOVA software versions.

The subscriber interface is more complex. Two peripheral devices are used (VIAs) and these devices have an array of registers accessible by BIU software. The subscriber interface is slightly different in the MODCOMP and NOVA BIU software.

The discussion will focus on the hardware interface components of the BIU, how they are used, and what software is required to invoke them. The reader is referred back to Figure 3.0-1 for identification of the overall structure of the BIU hardware components and the software names used for interface lines.

10.1 Network Interface Protocol

The 6502 interfaces to the net through a Motorola 6850 Asynchronous Communications Interface Adapter (ACIA- See [8]). The 6502 controls, monitors, and exchanges data with the ACIA via the ACIA's four, eight-bit-wide registers. These four registers are wired to the 6502 in such a way that they appear to be ordinary memory locations to the BIU software.

The 6502 controls and monitors the activities of the ACIA by setting the ACIA's control register and reading the ACIA's status register. Both the control register and the status register are referenced through address $0C00

(which has been named "NUARTS" in the software). A "LDA NUARTS" instruction moves the current contents of the ACIA's status register into the 6502's accumulator. A "STA NUARTS" instruction moves the contents of the 6502's accumulator into the ACIA's control register. The ACIA's control register cannot be read by the 6502. Likewise its status register cannot be altered (except as part of a master reset as described below).

The 6502 exchanges data with the ACIA via the ACIA's Transmit and Receive Data Registers. The ACIA likewise uses these two registers to exchange data with the network. Again, both of these registers are accessed through one address, $0C01, which has been named "NUARTD" in the software. A load from that location obtains data received from the network. A store into that address gives the ACIA data to be put on the network.

The meanings of the individual bits in the status and control registers (in the context of the BIU) are shown in Table 10.1-1. By changing the appropriate bits in the control register, the BIU can turn the ACIA's receiver on or off, tu. n the ACIA's transmitter on or off, and cause a master reset of the ACIA. A master reset clears the status register and effectively destroys any data in the receive or transmit data registers. In the BIU software, master resets are issued when a RESET or NMI interrupt occurs. The receiver and transmitter are turned on or off as deemed necessary in the routines RESET, NET, NMI, and NINT.

The ACIA's status register is read in the routines NET and IRQ to determine whether data can be written to or read from the net. The rule in NET is that a byte of data should be put in the transmit register only if the ACIA detects that the network is not busy and the ACIA has transmitted the last byte given to it. The rule in NINT is that there is meaningful data in the receive register only if the network is busy, the receive register is full, and the parity error indicator is off.

10.2 Subscriber Device Interface Protocol

The 6502 interfaces to the attached subscriber device (MODCOMP or NOVA) through two 6522 Versatile Interface Adapters (VIAs - See [9]).

Table 10.1-1
Contents of "NUARTS" - The ACIA's Status/Control Register

BIT	CONTENTS WHEN CONSIDERED AS A STATUS REGISTER (When read by 6502)	BIT	CONTENTS WHEN CONSIDERED AS A CONTROL REGISTER (When written by 6502)
7	1 if ACIA has an out-standing interrupt to the 6502.	7	1 Enable receiver 0 Disable receiver
6	1 if a parity error has been detected by the ACIA.	6-5	10 Enable transmitter 00 Disable transmitter
5-3	Not used in BIU	4-2	110 at all times
2	0 if the network is busy	1-0	11 Perform master reset of ACIA
1	1 if the network has taken the last byte stored into "NUARTD"		
0	0 if the network has loaded a byte into "NUARTD"		

One VIA is used for a 16-bit parallel bidirectional exchange of data with the subscriber. Transfers through this VIA require cooperation and coordination between the 6502 and subscriber device. Some of the required coordination is provided automatically through "handshaking" lines built into the VIA and connected to the subscriber device. These lines do not, however, provide enough information to support the required cooperation. Thus a second VIA is included in the BIU.

This second VIA is used to convey status signals rather than data between the 6502 and the subsriber device. Two sets of eight one-bit signal lines, one set operating in each direction, are available in this VIA. Fewer than eight, however, are actually used. This second VIA is physically identical to the first, but is used in a different manner.

The two different versions of the BIU (MODCOMP and NOVA) use the handshaking lines and the "status-transfer" VIA in a similar, though slightly different, manner to coordinate data transfers. Differences in software structuring mask some of the similarity between the two versions.

To accentuate the similarities between the MODCOMP and NOVA interface protocols, the protocols are explained below in terms of the operational characteristics of the VIA. The following sections will first describe usage of specific hardware features in the VIAs, including:

- usage of the data registers to transfer both data and status signals
- the "handshaking" signals available in the VIA,
- timers contained in the VIA, and
- control and flag resiters in the VIA and accessible by the 6502.

These explanations will be followed by simple "walk-throughs" detailing the sequence of events which must occur in both the subscriber and the 6502 to accomplish both a read and a write operation. This will be done for both the MODCOMP and NOVA versions of the protocol.

The various registers in the VIA are directly accessible by the 6502 in the same manner as the registers in the ACIA network interface device. That is, they are addressed as if they were ordinary memory locations and

are "loaded from" and "stored into" rather than "read" or "written". Table 10.2-I ties together some of the discussions of succeeding sections. The table lists the registers in each VIA which are referred to by the 6502 software. It also lists the variable name used in the software, the address to which the register is wired, and the manner of use of the register by the 6502. As mentioned above, the two VIAs are physically identical. The fact that fewer registers are referenced in the second VIA reflects not a difference in structure but a difference in use.

10.2.1 Data Registers

Each VIA contains two eight-bit data registers which can be accessed by both the 6502 and the subscriber device. At a given time, only one direction of data movement through a data register is permitted. That is, a data register is readable only by either the 6502 or the subscriber, and is writeable by the other. The allowable direction of movement of data through a data register is switchable and under the control of the 6502*. In the BIU, the two data registers in the "data transfer" VIA are switched back and forth together to form a 16-bit path in either direction.

The two data registers in the "status transfer" VIA are always used in a fixed direction. One referred to as "PORT2A" in the software, carries status signals from the subscriber device to the BIU. The other, "PORT2B", carries status signals to the subscriber from the BIU. The status signals used differ between the MODCOMP and NOVA versions of the BIU. Tables 10.2.1-I and 10.2.1-II present bit maps of the contents of "PORT2A" and "PORT2B" for both versions. Differences in the two versions reflect differences in the hardware and software characteristics of the subsribers and differences in the role of the BIU.

---

* Through the use of the Data Direction Register described below. Actually the direction of the Data Register can be set on a bit-by-bit basis rather than as an eight-bit unit. This, however, is not done in the BIU.

Table 10.2-I VIA Registers: Software Names, Addresses, and Usage by 6502

Register Type	VIA #1 (Data Transfer VIA)			VIA #2 (Status Transfer VIA)		
	Software Name	Address	Relation to 6502*	Software Name	Address	Relation to 6502*
Data Register B	"PORT1B"	$1010	R1/W1	"PORT2B"	$1020	W1
Data Register A	"PORT1A"**	$1011	R1	"PORT2A"	$1021	R1
Data Direction Register B	"P1BDDR"	$1012	W2	"P2BDDR"	$1022	W2
Data Direction Register A	"P1ADDR"	$1013	W2	"P2ADDR"	$1023	W2
TIMER 1	"TIMRL"	$1014	XX			
TIMER 1	"TIMRH"	$1015	XX			
TIMER 2	"TIM2L"	$1018	XX			
TIMER 2	"TIM2H"	$1019	XX			
Auxiliary Control Register	"P1ACR"	$101B	W2	"P2ACR"	$102B	W2
Peripheral Control Register	"P1PCR"	$101C	W2	"P2PCR"	$102C	W2
Interrupt Flag Register	"P1IFR"	$101D	R2			
Interrupt Enable Register	"P1IER"	$101E	W2	"P2IER"	$102E	W2
Data Register A	"P1ANHS"**	$101F	W1			

*R1 – Read by 6502, written by subscriber device
W1 – Written by 6502, read by subscriber device
R2 – Read by 6502, written by VIA based on I/O status
W2 – Written by 6502, used by VIA
XX – See discussion of timers in Section II

**PORT1A and P1ANHS both reference the same register. References to PORT1A, however, cause handshaking signals to be generated which are not generated by references to P1ANHS. See text for complete explanation.

Table 10.2.1-I Contents of "PORT2A" -
Status Signals Sent to BIU from the Subscriber Device

BIU Software Version	Bit	Software Name	Values	Comments
NOVA Version	7	"NBOOT"	0 NOVA contains executable program 1 NOVA does not contain executable program	Cleared by NOVA software when program loaded from disk or BIU Set when "reset" button on NOVA hit
	6	"NBUSY"	0 Interface board in NOVA has I/O underway 1 Interface board in NOVA does not have I/O underway	Stays clear until NOVA has read/written the last word of a transfer to the VIA, then is set to 1. May be set to 1 before 6502 reads last word from VIA.
	5	"NDONE"	0 Interface board in NOVA is attempting to interrupt NOVA to say it is done with I/O	Referred to by BIU only so that BIU doesn't attempt to generate an interrupt to the NOVA while a previous interrupt is outstanding
MODCOMP Version	7	"BUSYN"	0 MODCOMP has an outstanding I/O operation to BIU 1 MODCOMP does not have an outstanding I/O operation to BIU	MODCOMP can only be interrupted by BIU when it has a read outstanding. MODCOMP software thus almost always has at least a "dummy" read outstanding so BUSYN is almost always 0.

Table 10.2.1-II Contents of "PORT2B" -
Status Signals Sent to BIU from the Subscriber Device

BIU Software Version	Bit	Software Name	Values	Comments
BOTH	7	"NINTRP" (in NOVA) "EXTSIN" (in MODCOMP)	0 Interrupt of Subscriber Device being requested by BIU 1 Interrupt of Subscriber Device not being requested by BIU	Bit only needs to be cleared for one 6502 instruction cycle to generate interrupt. It should then be set again.
	6	"INRDY"	0 BIU does not have data ready for subscriber. 1 BIU has packet for subscriber	
	0	"INOUT"	0 VIA is set up for I/O from subscriber to BIU 1 VIA is set up for I/O from BIU to subscriber	Bit is not passed on to subscriber but controls supplementary switches in BIU.
NOVA Only	5	"OUTRDY"	0 BIU does not have a buffer available for msg. from NOVA 1 BIU has buffer available for msg. from NOVA	
	4	"ERROR"	0 No errors in BIU/ subscriber interface 1 Error detected by BIU	Interface software in NOVA does not currently process this bit.
MODCOMP Only	5	"TO"	0 Time out not detected by BIU 1 BIU timer has timed out	See Section XI
	4	"S3"	1	Must be set to 1 to keep MODCOMP interface board happy.

10.2.2 "Handshaking" Signal Lines

Each "port" (data register) in each VIA has associated with it two signal lines connected to the subscriber device. The signal lines are used for "handshaking" during data transfers between the 6502 and the subscriber device. One carries signals from the VIA to the subscriber, the other carries signals in the opposite direction. Thus each VIA has two signal lines running in each direction.

In the data transfer VIA, a signal is sent on one line (called CB2) when the 6502 writes into the data registers. The subscriber returns a signal on a second line (CB1) when it reads the data register. A third line (CA1) is used to signal that the subscriber has written to the data registers and the fourth line (CA2) signals a read by the 6502.

The 6502 does not directly generate any of these signals, nor does it directly sense them. The signals to the subscriber (CA2 and CB2) are generated automatically by the VIA when it detects that the 6502 has, respectively, read or written into Port 1A and Port 1B. Similarily the VIA receives the CA1 and CB1 signals from the subscriber and raises flags in the Interrupt Flag Register (IFR), which is periodically interrogated by the 6502. Thus these signals are nowhere explicitly referenced in the BIU software.

The signals are however implicitly referenced in that there are two ways of accessing the "A" data register in the data transfer VIA. The software locations "PORT1A" and "P1ANHS" both correspond to this data register. The difference between the two lies only in the handshaking signals which will be automatically generated by the VIA when the locations are accessed. A signal will be sent on CA2 when "PORT1A" is either written to or read from. On the other hand, neither type of reference to "P1ANHS" will generate a signal. In the BIU, the location "PORT1A" is used for all read accesses to the data register by the 6502, and "P1ANHS" is used for all write accesses. Thus a signal is sent on CA2 only when the 6502 reads the "A" data register.

A signal is sent on CB2 when PORT1B is written to by the 6502. A read of PORT1B generates no signal at all.

The status transfer VIA has the same four signal lines as does the data transfer VIA. The signals are not used, however, since there is no need in the BIU for coordinating the transfer of status signals. The status signals sent to the BIU in "PORT2A" are dynamically updated to reflect current interface conditions. The status signals sent to the subscriber in "PORT2B" are only read by the subscriber when it is interrupted by the BIU.

10.2.3 Timers

Each VIA has two separate timers which can be started and stopped under control of the 6502. In the BIU these timers are used to control how long the BIU will wait for certain events to occur. The use of the timers is only indirectly a part of the interface protocol and thus will not be discussed here. See Section XI for a discussion of timer usage.

10.2.4 Control and Interrupt Flag Registers

Each VIA also has several control registers which determine its operational characteristics and an Interrupt Flag Register which indicates its status. These registers are accessible by the 6502. Briefly, the control registers determine:

- the direction of data transfers,
- the operational characteristics of the timers,
- the nature of the "handshaking" signals to be sent to the subscriber when the 6502 reads and writes the data registers,
- the nature of the "handshaking" signals expected to be received from the subscriber device when it reads and writes the data registers, and
- which conditions, if any, detected by the VIA should generate an interrupt to the 6502.

The following subsections describe in detail the four types of control registers and the Interrupt Flag Register. In general the control registers are set only in the BIU's initialization phase (the RESET routine). Table 10.2.4-I indicates the values written to these registers in both the MODCOMP and NOVA versions. The following sections will indicate what effect these values have on the operation of the VIA and why there are differences between the two versions of the software.

Table 10.2.4-I
VIA Control Register Settings Used by Both Versions of BIU Software

VIA CONTROL REGISTER	VALUE IN NOVA VERSION OF BIU	VALUE IN MODCOMP VERSION OF BIU
P1ADDR	*	*
P1BDDR	*	*
P2ADDR	00000000	00000000
P2BDDR	11111111	11111111
P1ACR	01000000	01000000
P2ACR	00000000	00000000
P1PCR	10101010	10101010
P2PCR	10101010	00000000
P1IER	01111111	10100000
P2IER	01111111	01111111

* Value changed by software as required by I/O traffic. Possible values 00000000 or 11111111. P1ADDR and P1BDDR should both be set the same.

10.2.4.1 Data Direction Register (DDR)

Number: One per 8-bit port ⟶ two per VIA ⟶ four per BIU

Software names: "P1ADDR," "P1BDDR," "P2ADDR," "P2BDDR"

Function: Specifies in which direction data will move through the corresponding port (either from BIU to subscriber device, or vice versa). Each "bit" (line) in a port can be controlled individually.

Fields: Bits 7-0 Each bit specifies the direction of movement of data through the corresponding bit in the corresponding port:
0 - from subscriber to BIU
1 - from BIU to subscriber

Set by: 6502 software

Differences in use between MODCOMP and NOVA BIU versions: None.

Set to (values shown are binary numbers):

"P2ADDR" = 00000000 "P2BDDR" = 11111111	in RESET
"P1ADDR" = 00000000 "P1BDDR" = 00000000	in RESET and after termination of a write from the BIU to the subscriber device
"P1ADDR" = 11111111 "P1BDDR" = 11111111	at initiation of a write to the subscriber device

Comments: "PORT2A" is always used for eight bit transfers from the subscriber device to the BIU. "PORT2B" is always used to send information from the BIU to the subscriber device. (Both ports carry only status signals and not data. For more information see paragraph 10.2.1.)

"PORT1A" and "PORT1B" are used together as a 16-bit parallel patch between the BIU and the subscriber device. The normal direction of the path is from the device to the BIU. The direction is reversed when the BIU has something to write to the subscriber device and the device is willing to take it. The path is switched back to its normal direction after the output operation is terminated

for any reason. Attempted data transfers in a direction counter to that indicated by the DDR may either be ignored or generate bad data.

10.2.4.2 Auxiliary Control Register (ACR)

Number: One per VIA ⟶ two per BIU

Software Name: "P1ACR," "P2ACR"

Function: Specifies how the two timers and the shift register in the VIA are to be used; specifies (for each of the two ports in the VIA - A and B) whether signals are to be "latched" into the ports or allowed to change continously.

Fields: Bits 7-6 TIMER 1 characteristics
Bit 5 TIMER 2 characteristics
Bits 4-2 Shift register usage characteristics
Bits 1-0 Presence or absence of latching for ports A and B.

Set by: 6502 software

Differences in use between MODCOMP and NOVA versions: None

Set to (values shown are binary numbers):

P1ACR = 01000000
P2ACR = 00000000

The ACRs are modified by the 6502 software only in the RESET routine. They are not referenced at any other time.

Comments: TIMER 1 in the first VIA is defined to be a "free-running" timer. TIMER 2 in the first VIA and both timers in the second VIA are defined to be interval timers operating in a one-shot mode. For more information on the uses of these timers and the distinctions between timer types see Section XI.

The shift register in the VIA is not used in the BIU.

No latching is used in the BIU. Thus a "read" from a data register will obtain the current value of the input lines attached to it, rather than the first detected change on those lines. Thus inputs from the subscriber device can theoretically be overwritten and lost if not read in time. Auxiliary handshaking procedures, however, should prevent two consecutive writes to the register without an intervening read.

10.2.4.3 Peripheral Control Register (PCR)

Number: One per VIA ——> to per BIU

Software Names: "P1PCR," "P2PCR"

Function: Specifies how the CA1, CA2, CB1, CB2 lines are to be used in "handshaking" protocol with the subscriber device.

Fields:

Bits 7-5	CB2	signal characteristics
Bit 4	CB1	signal characteristics
Bits 3-1	CA2	signal characteristics
Bit 0	CA1	signal characteristics

Set by: 6502 software

Differences in use between MODCOMP and NOVA BIU versions: The PCR for the "data transfer" VIA is used in an identical manner in both versions. The PCR for the "status transfer" VIA is set differently in the two versions to correspond to differences in the interface boards used in the MODCOMP and NOVA. This difference in the use of the PCR does not affect any other aspect of the BIU software.

Set to (values shown are binary numbers):

P1PCR = 10101010 in both versions
P2PCR = 10101010 in NOVA version only
P2PCR = 00000000 in MODCOMP version only

The PCRs are modified by the 6502 software only in the RESET routine. They are not referenced at any other time.

Comments: The CA1 and CB1 lines carry signals from the subscriber device to the 6522. The 6522 monitors these lines looking for a transition. The transition it is looking for is specified by the PCR. In the case of the BIU, the PCR specifies a high-to-low transition to reflect the nature of the circuitry in the interface board in the subscriber device. When the appropriate transition is found, the corresponding bit in the interrupt flag register is raised.

The CA2 and CB2 lines carry signals from the 6522 to the subscriber device. The '101' value loaded into "P1PCR" for controlling these two lines in the "data transfer" VIA will cause a signal to be sent on CA2 each time Port A is read or written by the 6502, and a signal to be sent on CB2 only when Port B is written.

The same signals will be sent on CA2 and CB2, in the NOVA version, for accesses to the data registers in the status transfer VIA. In the MODCOMP version, however, the '000' loaded into "P2PCR" causes no signals to be sent on these two lines.

For more information on the use of CA1, CB1, CA2, and CB2 see paragraph 10.2.2.

10.2.4.4 Interrupt Enable Register (IER)

Number: One per VIA ⟶ two per BIU

Software Names: "P1IER," "P2IER"

Function: Specifies which conditions detected by the VIA shall cause interrupts to the 6502. The VIA is constantly looking for certain conditions to occur. When one of those conditions occurs, a bit in the Interrupt Flag Register is set. If the corresponding bit in the Interrupt Enable Register is set, an IRQ interrupt is sent to the 6502.

Fields: Correspond to the fields in the Interrupt Flag Register.

Set by: 6502 software

Differences in use between MODCOMP and NOVA versions: "P1IER" is set differently to reflect differences in the software. The MODCOMP BIU uses two timers, while the NOVA BIU uses only one. The second timer is required to account for an unusual characteristic of the MODCOMP described in Section XI.

Set to (values shown are binary numbers):

P1IER = 01111111 in the NOVA version
P1IER = 10100000 in the MODCOMP version
P2IER = 01111111 in both versions

The IERs are modified by the 6502 software only in the RESET routine. They are not referenced at any other time.

Comments: In the NOVA version, all interrupt types are disabled. No interrupts to the BIU will be generated by either VIA. The 6502 will monitor the status of interrupt conditions by periodically reading the Interrupt Flag Register.

In the MODCOMP version, all interrupt types but one (interrupts generated by TIMER 2 in the first VIA) are disabled. Section XI discusses when this interrupt will occur and how it will be handled.

If bit 7 of the data written to the IER is a 0, each 1 in bits 6 through 0 clears the corresponding bit in the IER. For each 0 in bit 6 through 0, the corresponding bit is unaffected. Setting selected bits in the IER is accomplished by writing to it with bit 7 set to a logic 1. In this case, each 1 in bits 6 through 0 will set the corresponding bit. For each 0, the corresponding bit will be unaffected.

10.2.4.5 Interrupt Flag Register (IFR)

Number: One per VIA ⟶ two per BIU (however only one is used)

Software Name: "P1IFR"

Function: Inform the 6502 of the status of the VIA (and thus, indirectly, of the subscriber device).

Fields:

Bit 7	not used in BIU
Bit 6	"TOFLAG" (TIMER 1 status)
Bit 5	not used in BIU
Bit 4	"OUTFLG" (status of output to subscriber device)
Bits 3–2	not used in BIU
Bit 1	"INFLAG" (status of input from subscriber device)
Bit 0	not used in BIU

Set by: VIA according to timer status and the VIA's interactions with the subscriber device.

Differences in use between MODCOMP and NOVA versions: None.

Set to:

"TOFLAG"
- set to 1 when TIMER 1 has timed out
- reset to 0 when TIMER 1 is reset (see Section XI)

"OUTFLG"
- reset to 0 when 6502 writes to (or reads from) PORT1B
- set to 1 when subscriber device has read data from both PORT1B and PORT1A

"INFLAG"
- set to 1 when subscriber device has written data into PORT1B and PORT1A
- reset to 0 when 6502 has taken data from PORT1A

Comments: "P1IFR" is read by the 6502 and used in conjunction with the status signals transferred through "PORT2A" to determine the subscriber device's I/O status. The meaning of the bits in "P1IFR" is the same for both the MODCOMP and NOVA versions. However the signals in "PORT2A" differ between the two versions and the definition of subscriber device states and interface protocols differ slightly.

It should be noted that since these flags indicate past signal transitions only, they do not indicate the levels of the signal lines with which they are associated. Furthermore, clearing a bit in the IFR does not affect the state of the line associated with the bit.

10.2.5 Sample Data Transfers Between the BIU and the Subscriber Device

Figures 10.2.5-1 and 10.2.5-2 trace the sequence of steps required of both the BIU and the subscriber device to accomplish data transfers (excluding the setting of timers which are used to detect when an operation takes too much time - See Section XI). The figure indicates the direct actions taken in the BIU software, the implicit activities generated in the VIA, and the actions of the subscriber device. The figure shows how the status bits in "PORT2A," "PORT2B" and "P1IFR" are changed as the transfer progresses.

The scenario portrayed by the figures assumes that no I/O is in progress at the start of the current operation and that no I/O immediately follows. Had there been ongoing or follow-up I/O, the sequence of instructions would remain the same, however some queue processing and priority calculations would be carried out before and after the transaction.

	MODCOMP INTERFACE ACTIVITY	NOVA INTERFACE ACTIVITY
Normal Idle State	INRDY ≡ 0 EXTSIN ≡ 1 Outstanding "Dummy" read BUSYN ≡ 0 OUTFLG ≡ indeterminate INFLAG ≡ 0	OUTRDY ≡ 1 INRDY ≡ 0 NINTRP ≡ 1 NBUSY ≡ 1 OUTFLG ≡ indeterminate INFLAG ≡ 0
Interrupt Subscriber to Advise of Waiting Data	INRDY ⟵ 1 EXTSIN ⟵ 0 EXTSIN ⟵ 1 BUSYN ⟸ 1	INRDY ⟵ 1 NINTRP ⟵ 0 NINTRP ⟵ 1
Subscriber issues read operation	BUSYN ⟸ 0	NBUSY ⟸ 0
Converts Port 1 to an output port	P1ADDR ⟵ 11111111 P1BDDR ⟵ 11111111 INOUT ⟵ 1	P1ADDR ⟵ 11111111 P1BDDR ⟵ 11111111 INOUT ⟵ 1
Loop, transferring data	P1ANHS ⟵ data PORT1B ⟵ data OUTFLG ⟵ 0 (handshake) [data taken by subscriber device] OUTFLG ⟸ 1 (handshake)	P1ANHS ⟵ data PORT1B ⟵ data OUTFLG ⟵ 0 (handshake) [data taken by subscriber device] OUTFLG ⟸ 1 (handshake)
At end of transfer, reset Port 1 to be an input port	P1ADDR ⟵ 00000000 P1BDDR ⟵ 00000000 INOUT ⟵ 0	P1ADDR ⟵ 00000000 P1BDDR ⟵ 00000000 INOUT ⟵ 0
Interrupt subscriber to signal end of operation	INRDY ⟵ 0 EXTSIN ⟵ 0 EXTSIN ⟵ 1	INRDY ⟵ 0 NINTRP ⟵ 0 NINTRP ⟵ 1
Subscriber ends operation	BUSYN ⟸ 1	NBUSY ⟸ 1

Figure 10.2.5-1 General Steps Involved in Write by BIU to Subscriber Device

Notation: ⟵ means transfer accomplished by BIU
⟸ means transfer accomplished by Subscriber

	MODCOMP INTERFACE ACTIVITY	NOVA INTERFACE ACTIVITY
Normal Idle State	INRDY ≡ 0 EXTSIN ≡ 1 BUSYN ≡ 0 OUTFLG ≡ 1 INFLAG ≡ 0 Outstanding "Dummy" read	OUTRDY ≡ 1 INRDY ≡ 0 NINTRP ≡ 1 NBUSY ≡ 1 OUTFLG ≡ 0 INFLAG ≡ 0
Subscriber issues write operation	BUSYN ⟸ 1 BUSYN ⟸ 0	NBUSY ⟸ 0
Loop, transferring data	data made available by subscriber device INFLAG ⟸ 1 (handshake) PORT1B ⟶ BIU memory PORT1A ⟶ BIU memory INFLAG ⟵ 0 (handshake) OUTFLG ⟵ 0 (consequence of handshake)	data made available by subscriber device INFLAG ⟸ 1 (handshake) PORT1B ⟶ BIU memory PORT1A ⟶ BIU memory INFLAG ⟵ 0 (handshake) OUTFLG ⟵ 0 (consequence of handshake)
Subscriber ends operation	BUSYN ⟸ 1	NBUSY ⟸ 1

Figure 10.2.5-2
General Steps Involved in Read by BIU from Subscriber Device

Notation: ⟵ means transfer accomplished by BIU
⟸ means transfer accomplished by subscriber

SECTION XI
TIMERS AND CLOCKS

11.0 INTRODUCTION

Various processes in the BIU software are time dependent. These include:

- generation of status messages,
- the waiting period for a sign-on acknowledgement packet [NOVA version only],
- generation of interrupts to the subscriber device [MODCOMP version only],
- the waiting period for acknowledgement of packet reception, and
- the waiting periods between packet retransmissions due to collisions and lack of buffer storage in the destination BIU.

The following two sections briefly describe the hardware timing devices in the BIU and the manner in which they are used by the software.

11.1 Hardware Timers

There are two separate timers in each VIA (called TIMER 1 and TIMER 2) and thus four in each BIU. The MODCOMP version of the BIU software, however, uses only two of the four TIMER 1 and TIMER 2 in VIA 1) and the NOVA version uses only one TIMER 1 in VIA 1). The use and operation of TIMER 1 is identical in both versions. The operation of TIMER 2 is somewhat different from that of TIMER 1.

TIMER 1 is used in the BIU in what is termed a "free-running mode." That is, the period of the timer (T_1 clock cycles) is set during system initialization, and the timer is started. The timer will raise an interrupt flag at the end of the period and then restart itself with the initially specified period.

The timer does not have to wait to be restarted by an interrupt handler. Thus, the interrupt flag is raised precisely every T_1 clock cycles.

TIMER 2 operates in a "one-shot" mode. That is, after the initially specified period (T_2 clock cycles) expires and an interrupt flag is raised, the timer must be explicitly restarted by an interrupt handling routine. A finite variable amount of time is required to invoke the interrupt handler and reset the timer for another T_2 clock cycles. This will produce a period of slightly greater than T_2 clock cycles between timeouts for TIMER 2.

The interrupt flags raised by both TIMERs 1 and 2 cannot be directly reset. They can only be reset as a side effect of reading from or writing to specific fields of the timer's period and current value. Explanations of how each is reset in the BIU are given below. The interrupt flags for TIMERs 1 and 2 are in the Interrupt Flag Register of the VIA. As discussed in Section 10.2, raising of an interrupt flag may or may not (depending on the Interrupt Enable Register) cause an interrupt to the CPU. In the BIU, the expiration of TIMER 1 will not generate an interrupt to the CPU, but the expiration of TIMER 2 will.

In the terminology of the BIU software, TIMER 1 is initially started (in RESET) when a period for it is loaded into the 16-bit register pair "TIMRL" and "TIMRH." ("TIMRL" must be loaded first.) This load also clears the bit "TOFLAG" in "P1IFR." Expiration of TIMER 1 sets the bit "TOFLAG" to 1.

"TOFLAG" is polled by the routine TIMOUT whenever it is executed. If the bit is found to be set, a software timer (see next section) will be modified and the bit will be reset. TIMER 1 automatically restarts itself when it sets "TOFLAG.") The "TOFLAG" is reset as a side effect of reading the lower half of the clock period (i.e., by the "LDA TIMRL" instruction). In TIMOUT, the value of "TIMRL" is irrelevant (it is already known) but it is read to produce the side effect of resetting "TOFLAG."

TIMER 2 is also initially started in RESET by loading a 16-bit register pair "TIM2L" and "TIM2H." (Again "TIM2L" must be loaded first.) When the timer expires, an interrupt flag is raised and (if the interrupt enable bit is set as it is in the MODCOMP version) an interrupt to the 6502 is generated. The routine IRQ is then immediately invoked. IRQ first determines that the interrupt occurred because TIMER 2 expired (note: IRQ is also invoked if an interrupt is received from the network). Then the interrupt flag is cleared and the timer restarted by resetting the upper half of the timer's period ("TIM2H"). IRQ then modifies a software timer (see Section 11.2). The interrupt flag for TIMER 2 is in "P1IFR" but is not explicitly named since it is never read and never directly modified.

11.2 Software Clocks, Timers, and Event Scheduling

The two hardware timers discussed in the previous section are, indirectly, the basis of almost* all timing activities in both versions of the BIU software.

A current "time-of-day" clock is maintained in the variable "TOD" using TIMER 1. "TOD" expresses the time since the last reset of the BIU. The unit of time used in the MODCOMP is the second (approximately) and in the NOVA it is the quarter-second.

"TOD" is a three-byte variable and is maintained by the TIMOUT routine. TIMOUT counts the number of times hardware TIMER 1 expires. On every 100th expiration (in the MODCOMP version, 25th in the NOVA), "TOD" is incremented by 1. The variable "TICK" is used to count expirations of TIMER 1.

The "TOD" clock is used in conjunction with the STIMER and CTIMER routines to schedule future events and to decide if it is time yet for a scheduled event to occur. For example, consider the generation and transmission of

*The exception is in the NET routine. In NET, timers are simulated by looping through a short set of instructions. The length of such a wait or delay can be estimated by counting the number of instructions executed and multiplying by the 6502's basic instruction time of 600 nanoseconds.

BIU status messages onto the net. A status message is to be sent out every 60 seconds, with the first message occurring 60 seconds after a "reset" of the BIU.

In the RESET routine the three byte variable "TSTAT" is set to 60 seconds (or 240 quarter-seconds in the NOVA version). The routine CKTOUT will, whenever it is executed, compare the current time "TOD" to the scheduled time "TSTAT" using the CTIMER subroutine. When it is determined that "TOD" has exceeded "TSTAT," the message is sent and "TSTAT" is reset to "TOD" plus 60 seconds using the STIMER subroutine.

In the NOVA version, the period for waiting for an acknowledgement to a sign-on request is measured in much the same way as is the period for sending a status message. The variable "TSACK" is set to "TOD" plus maximum wait time. If a sign-on acknowledgement has not been received before this time, the system is considered unavailable. "TSACK" is set by SPMSGO invoking STIMER, and is monitored by CKTOUT invoking CTIMER.

The variable "TIMER" is used in the MODCOMP version of the BIU to decide when a "wake up" interrupt should be sent to the MODCOMP. A "wake-up" interrupt should be sent under two conditions. First, the MODCOMP should be notified if an I/O operation between it and the BIU takes "too" long (to be defined below). It is assumed that if this happens, something in the transfer has gone wrong. It is furthermore hoped that interrupting the MODCOMP may straighten things out. Second, the MODCOMP/BIU protocol specifies that the MODCOMP be interrupted about every 0.1 second during idle periods as a means of communicating that the BIU is still operating normally.

The variable "TIMER," unlike the variables discussed above, operates as a timer. That is, it is set to a desired interval length, is "automatically" decremented, and indicates that it is time for an event to occur when it expires (reaches 0). "TIMER" is set to 0.02 seconds at the

beginning of each write to the MODCOMP (maximum of 128 bytes) in PUTM and WRITEM. "TIMER" is set to 0.05 seconds at the beginning of each read from the MODCOMP (maximum of 1024 bytes) in GETM. "TIMER" is set to 0.10 seconds in RESET, after the termination of an I/O in PUTM and GETM, and after it has expired (in EXPTIM). "TIMER" is monitored by PUTM, GETM, WRITEM, READM, and CKTOUT. If any of these routines detect a timeout, the routine EXPTIM is invoked to generate the interrupt and reset "TIMER."

"TIMER" is based on the hardware TIMER 2 and is maintained by the IRQ routine. "TIMER" is decremented every time TIMER 2 expires and sends an interrupt to the 6502. IRQ is responsible for resetting hardware TIMER 2.

More information concerning the choice of timer intervals and the details of interface protocols is found in [3].

REFERENCES

[1] Brown, J.S., Trend Monitoring System (TMS) Graphics Software, The MITRE Corporation, MTR-4725 (JSC #14795), March 1979.

[2] Brown, J. S. and Weinrich, S. S. , Trend Monitoring System (TMS) Communications Hardware - Volume I - Computer Interfaces, The MITRE Corporation, MTR-4721 (JSC #14682), February 1979.

[3] Brown, J. S. and Lenker, M.D., Trend Monitoring System (TMS) Communications Software - Volume I - Computer Interfaces, The MITRE Corporation, MTR-4723 (JSC #14792), April 1979.

[4] Brown, J. S. and Hopkins, G. T., Trend Monitoring System (TMS) Communications Hardware - Volume II - Bus Interface Units, The MITRE Corporation, MTR-4721 (JSC #14723), March 1979.

[5] Allen, M. A., The NASA Bus Communications Listening Device Software, The MITRE Corporation, MTR-4729 (JSC #16054), July 1979.

[6] Brown, J. S., Letter to C. G. Krpec, Subject: Source Programs for Bus Interface Units (BIUs) for the Trend Monitoring System (TMS), The MITRE Corporation, D72-L-423/HO, 22 June 1979.

[7] MCS6500 Microcomputer Family Programming Manual, MOS Technology, Inc., Norristown, Pennsylvania, January 1976.

[8] "MC 6850 Asynchronous Communications Interface Adapter (ACIA) Data Sheet," MOTOROLA Semiconductor Products, Inc., Phoenix, Arizona.

[9] "MCS 6522 Versatile Interface Adapter Data Sheet," MOS Technology, Inc., Norristown, Pennsylvania, November, 1977.